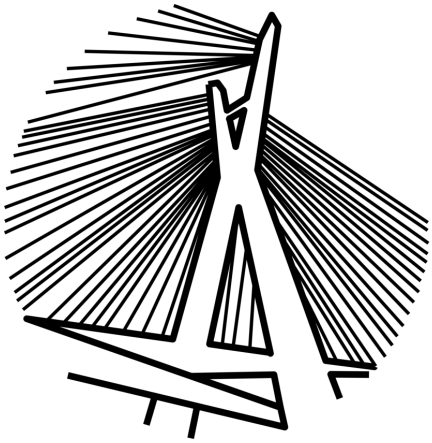
SBSeg 2019

MINICURSOS

XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais

2 A 5 DE SETEMBRO - SÃO PAULO - SP



SBSeg 2019

XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais

2 a 5 de Setembro – São Paulo, SP

MINICURSOS

Editora

Sociedade Brasileira de Computação – SBC

Organizadores

Marco A. A. Henriques, FEEC-Unicamp

Routo Terada, IME-USP

Daniel Macêdo Batista, IME-USP

Realização

Sociedade Brasileira de Computação – SBC

Instituto de Matemática e Estatística, Universidade de São Paulo – IME/USP

Promoção

Sociedade Brasileira de Computação – SBC

FICHA CATALOGRÁFICA

Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, (19º : 2019 : São Paulo, SP, Br)

S612 Minicursos do XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais - SBSeg 2019, [realizado em] São Paulo, SP, Brasil, [de] 02 a 05 de Setembro de 2019 / [orgs.] Marco A. A. Henriques ...[et al.] - São Paulo : Sociedade Brasileira de Computação, 2019.

180 p.

ISBN: 978-65-87003-89-4

1. Segurança de Redes. 2. Segurança de Computadores. 3. Redes e Comunicação de Dados. 4. Ciência da Computação. I. Henriques, Marco A. A., org. II. Terada, Routo, org. III. Batista, Daniel Macêdo, org. IV. Sociedade Brasileira de Computação. V. Instituto de Matemática e Estatística. Universidade de São Paulo. VI. Título.

Elaborada pelo Serviço de Informação e Biblioteca Carlos Benjamin de Lyra do IME-USP, pela bibliotecária Maria Lucia Ribeiro CRB-8/2766

Capa (Foto): Retirada de Wikipédia
https://pt.wikipedia.org/wiki/Ficheiro:Ponte_estaiada_Octavio_Frias_-_Sao_Paulo.jpg
Acesso em: 26/08/2019
Capa (Edição): Kétly Gonçalves Machado, IME-USP

Editoração: Kétly Gonçalves Machado, IME-USP
Daniel Macêdo Batista, IME-USP
Routo Terada, IME-USP

Sociedade Brasileira de Computação – SBC

Contato
Av. Bento Gonçalves, 9500. Setor 4, Prédio 43.412, Sala 219. Bairro Agronomia.
Caixa Postal 15012. CEP 91501-970. Porto Alegre - RS.

CNPJ: 29.532.264/0001-78
https://www.sbc.org.br

Comissão Especial de Segurança da Informação e de Sistemas Computacionais – CESeg

Coordenação

Altair Olivo Santin (PUCPR), Coordenador

Michele Nogueira (UFPR), Vice-Coordenadora

Comitê Gestor

André Luiz Almeida Marins (RNP)

Carlos Eduardo da Silva (UFRN)

Daniel Macêdo Batista (USP)

Eduardo Luzeiro Feitosa (UFAM)

Jean Everson Martina (UFSC)

Luciano Paschoal Gaspary (UFRGS)

Marcos Antonio Simplicio Junior (USP)

Paulo Lício de Geus (UNICAMP)

Rafael Timoteo De Sousa Junior (UnB)

Mensagem da Coordenação de Minicursos

O Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg) é o principal fórum nacional para a apresentação de resultados de pesquisas e atividades relevantes ligadas à segurança da informação, de protocolos de dados e de sistemas. É um evento científico promovido anualmente pela Sociedade Brasileira de Computação (SBC) que promove a integração da comunidade brasileira de pesquisadores e profissionais atuantes nessa área. Entre as diversas atividades técnicas do SBSeg encontra-se a trilha de Minicursos, que tem como objetivo atualizar os participantes em temas normalmente não cobertos nas grades curriculares dos cursos de graduação e pós-graduação e que despertam forte interesse entre acadêmicos e profissionais.

Nesta edição do SBSeg (2019), 13 propostas de minicursos foram submetidas, um número significativo que demonstra a relevância e o interesse da comunidade em temas atuais da área. Dentre estas propostas, quatro foram selecionadas para publicação e apresentação, representando assim uma taxa de aceitação de 30%. O Comitê de Programa dos Minicursos foi composto por renomados pesquisadores da área de segurança computacional para a elaboração dos pareceres. Cada proposta recebeu ao menos três pareceres e várias mensagens foram trocadas entre os membros do comitê durante a fase de discussão para se buscar uma convergência em relação aos quatro trabalhos que deveriam ser escolhidos, tarefa difícil dada a alta qualidade das propostas.

Este livro reúne os quatro capítulos produzidos pelos autores das propostas de minicursos aceitas. O Capítulo 1 trata de um dos grandes desafios para a segurança da informação na atualidade: a criação e manutenção confiáveis de uma identidade digital para cada usuário. Com o advento dos blockchains, novas possibilidades de atribuição e controle de identidades digitais surgiram e este minicurso busca trazer para os participantes uma discussão teórica e atividades práticas sobre emissão e gerenciamento de identificadores descentralizados, além de autenticação e revogação de credenciais.

O Capítulo 2 busca mostrar como a área de segurança pode se beneficiar dos enormes avanços obtidos recentemente nas áreas de ciência de dados e aprendizagem de máquina. O objetivo é conseguir descobrir informações relevantes do ponto de vista da segurança que podem estar ocultas em meio a uma grande quantidade de dados não suspeitos produzidos por ferramentas de segurança e pelos sistemas operacionais. Os conceitos-chave da área serão discutidos e vários exemplos práticos serão demonstrados com o uso de ferramentas gratuitas.

O Capítulo 3 traz uma abordagem mais profunda sobre blockchains, visto que esta ferramenta está se tornando cada vez mais utilizada em diversas áreas, inclusive a da segurança. Dentre as diversas tecnologias adotadas em blockchains, este minicurso dá ênfase ao mecanismo

de consenso, um dos principais atores nos quesitos desempenho e segurança da cadeia. Os mecanismos de consenso mais utilizados atualmente exigem um grande poder computacional para operarem satisfatoriamente e proverem um nível de segurança aceitável. Neste contexto, serão apresentados e discutidos os principais tipos de blockchain e seus mecanismos de consenso, bem como experimentos que ilustram na prática diferentes aspectos de segurança nas implementações atuais.

Concluindo este livro, o Capítulo 4 põe o foco em outra face da segurança que tem chamado bastante a atenção da sociedade: trata-se das aplicações maliciosas que, após infectarem um dispositivo, buscam se ocultar das mais variadas formas a fim de escaparem da detecção por mecanismos de segurança e por perícias forenses, mesmo quando estão em execução. Este minicurso mostra a aplicação de técnicas de engenharia reversa para análise de aplicações maliciosas no ambiente Linux, as quais são exemplificadas tanto nos modos kernel como usuário com base em seu traço dinâmico de execução e em técnicas de depuração de código. Nota-se portanto que tanto este como os demais capítulos trazem temas que têm despertado grande interesse recentemente na academia e na indústria e devem atrair um grande número de participantes.

Como Coordenador de Minicursos, gostaria de expressar o meu agradecimento aos membros do Comitê de Programa dos Minicursos por terem aceitado participar voluntariamente dessa empreitada e pelo excelente trabalho que fizeram no processo de avaliação e seleção dos minicursos. Gostaria de também agradecer aos Coordenadores Gerais do SBSeg 2019, Profs. Routo Terada (IME-USP) e Daniel Macêdo Batista (IME-USP), pela disponibilidade e suporte providos ao longo de todo o processo e pela confiança em nos delegar a coordenação dos minicursos. Finalmente, gostaria de agradecer aos autores por terem prestigiado este evento ao submeterem suas propostas.

Nossos votos de um excelente simpósio e um ótimo aproveitamento dos minicursos pelos participantes.

Marco Aurélio Amaral Henriques (Unicamp)
Coordenador de Minicursos do SBSeg 2019

Comitês

Comitê de Organização

Coordenação Geral
Routo Terada, USP
Daniel Macêdo Batista, USP

Coordenação de Minicursos
Marco Aurélio Amaral Henriques, Unicamp

Comitê de Programa dos Minicursos

Altair Santin, PUCPR
Dorgival Guedes, UFMG
Edna Canedo, UnB
Eduardo Souto, UFAM
Julio Hernandez, Unicamp
Luis Kowada, UFF
Marco A. A. Henriques, Unicamp
Rossana Andrade, UFC

Revisores

Altair Santin, PUCPR
Dorgival Guedes, UFMG
Edna Canedo, UnB
Eduardo Souto, UFAM
Julio Hernandez, Unicamp
Luis Kowada, UFF
Marco A. A. Henriques, Unicamp
Rossana Andrade, UFC

Sumário

Capítulo 1

Identidade Digital Descentralizada: Conceitos, aplicações, iniciativas, plataforma de desenvolvimento e implementação de caso de uso.

Emilio Tissato Nakamura[1], Fernando Cezar Herédia Marino[1], José Reynaldo Formigoni Filho[1], Sérgio Luís Ribeiro[1] e Vítor Padilha de Oliveira[1]

[1]CPQD
Rua Ricardo Benetton, 1000, Campinas, SP – Brasil

Abstract

This short course aims to demonstrate a conceptual and practical view of Decentralized Digital Identity of things and people based on blockchain. We will present the main concepts related to blockchain, digital identity and self-sovereign digital identity, global initiatives, as well as legal and standardization aspects. The hands-on part will include the creation of a DLT network specialized for creating and managing Decentralized Digital Identities as well as the development of an application that executes the main methods and routines of the DLT (Hyperledger Indy) framework, such as: issuing decentralized identifiers ("DIDs") between applications ("Agents"), managing these identifiers, proof authentication and revocation of credentials by the issuer.

Resumo

Esse minicurso tem o objetivo apresentar uma visão conceitual e prática de Identidade Digital Descentralizada, de coisas e pessoas baseada em blockchain. Para isso, serão apresentados os principais conceitos relacionados a blockchain, identidade digital e identidade digital autossoberana, iniciativas globais, assim como os aspectos legais e de padronização. A parte prática do minicurso contemplará a criação de uma rede DLT, especializada para a criação e gestão de Identidades Digitais Descentralizadas, assim como o desenvolvimento de uma aplicação que execute os principais métodos e rotinas do framework DLT (Hyperledger Indy), como por exemplo, emissão de identificadores descentralizados ("DIDs") entre aplicações ("Agents"), gerenciamento desses identificadores, autenticação de provas e a revogação de credenciais pelo seu emissor.

1.1 Introdução

Quando alcançamos a entrada do período Neolítico, há cerca de dez mil anos, os grupos humanos existentes já acumulavam um variado leque de saberes apreendidos graças à sua habilidade de raciocínio. Foi nesse contexto que uma profunda transformação passou a se desenvolver no cotidiano do homem pré-histórico, surgindo assim as primeiras técnicas de cultivo agrícola, garantindo alimento sem a necessidade de deslocamento, criando as primeiras comunidades. Essa transformação, que se difundiu ao longo dos próximos seis mil anos, deu origem à chamada "Revolução Neolítica ou Revolução Agrícola" [1].

Desde então, começamos a seguir e evoluir baseado no modelo centralizado – que está perpetuado em nossas mentes – onde, teoricamente, uma pessoa ou governo tomam decisões (por consenso centralizado) para beneficiar a comunidade e a população como um todo, nesse sentido a centralização tem sido um princípio organizacional básico para a economia e a sociedade desde então.

Organizações maiores com mais clientes tendem a usar integração vertical e funções centralizadas para aumentar a eficiência e reduzir os custos [2]. Observa-se que a centralização econômica é o princípio organizacional mais eficaz desde que, os custos de comunicação e transação sejam elevados[2]. Porém, notamos que esse modelo centralizado está mudando e trará inúmeras implicações para a humanidade como um todo. Pois, a Internet diminuiu os custos da comunicação, a qual tende a zero, e a tecnologia *blockchain* fará o mesmo para os custos da transação [3] quebrando assim, o paradigma do modelo centralizado de eficiência e redução de custos.

Acredita-se que a tecnologia *blockchain* tem o potencial de ser a força motora que irá democratizar a economia mundial, e com certeza, será considerada uma das tecnologias mais importante na história do século. Pois pelo modelo desenvolvido, nenhuma autoridade central é necessária, criando assim, a maior quebra de paradigma na qual precisamos nos habituar e entender, pois o método de consenso será decentralizado [4]. Além disso, a arquitetura de confiança empregada na tecnologia também é descentralizada, o que a torna mais eficiente, pois aproxima o cliente final e o vendedor sem intermediários. Nos processos centralizados, o modelo de confiança é baseado em Leis, o que normalmente cria barreiras e/ou proteção de mercado à entrada de novos participantes.

A premissa atual é que governo e tecnologia num futuro próximo serão uma coisa só e a atividade de governar passará necessariamente pelo uso da tecnologia. A consequência disso é que um governo que não evolui tecnologicamente deixa de ser governo tornando-se obsoleto e ineficaz. Alinhado a esse tipo de ação, temos dois exemplos, o primeiro é da Estônia, um país case no uso de tecnologia e principalmente identidade digital e *blockchain*.

A Estônia implementou um pioneiro sistema de identidades digitais [5], com o objetivo de fazer com que tudo o que o cidadão precise do Estado possa ser feito de forma digital. Uma pessoa só precisa ir pessoalmente a um órgão público em três casos: (i) casar, (ii) divorciar , e (iii) para vender um imóvel.

Outro exemplo é o da Índia, que também criou uma identidade digital – Aadhaar [6] para os seus 1,2 bilhões de habitantes. A identidade digital indiana se organiza em torno de quatro princípios: (i) dispensa de presença física, (ii) dispensa de papel, (iii) digitalização do dinheiro e consentimento.

Esses princípios significam que os indianos atualmente podem acessar todos os serviços públicos pelo celular e sem precisar de nenhum papel. Além disso, o sistema

protege a privacidade, permitindo que o cidadão decida qual dado pode ser acessado por qual órgão (e somente por ele). Tudo isso, como consequência, levou a um processo de intensa bancarização da população, permitindo a digitalização da economia e a redução da circulação física de dinheiro.

1.2 Conceitos da Tecnologia *Blockchain*

A tecnologia *blockchain* pode ser entendida de várias formas. Em linhas gerais, pode-se dizer que se trata de um sistema distribuído de base de dados em log, mantido e gerido de forma compartilhada e descentralizada (através de uma rede *peer-to-peer* - P2P), na qual todos os participantes são responsáveis por armazenar e manter a base de dados.

A tecnologia foi construída tendo em mente quatro principais características arquiteturais: (i) segurança das operações, (ii) descentralização de armazenamento e computação, (iii) integridade de dados e (iv) imutabilidade de transações.

Dito de outra forma, *blockchain* é uma "*ledger of facts*" replicada em computadores que participam de uma rede *peer-to-peer*, onde [7]:

- O *ledger* é um livro de registros digital, no qual uma vez validado um registro, este nunca mais poderá ser apagado;
- Um fato (*fact*) pode significar várias coisas, desde uma transação monetária, a um conteúdo de determinado documento, ou até mesmo um programa de computador, contendo, em algumas plataformas, até uma base de dados pequena;
- Os membros participantes da rede podem, ou não ser anônimos e são chamados *peers* ou "nós";
- Toda operação ou transação dentro da *ledger* é protegida por tecnologias criptográficas de assinatura digital, inclusive para identificar os nós emissores e receptores das transações;
- Quando um nó deseja adicionar ao *ledger* um fato novo, é necessário um consenso entre todos ou alguns nós previamente determinados da rede, para decidir se um fato pode ser registrado no *ledger*;
- Havendo consenso, o fato será escrito e nunca mais poderá ser apagado, em tese, um processo levemente semelhante à escritura e registro de um imóvel no Brasil.

Conforme apresentado na Figura 1, uma rede *blockchain* possui os seguintes elementos essenciais:

- Fato (*Fact*): pode ser uma transação, um conteúdo digital ou um programa de computador, este último também denominado contrato inteligente (*smart contract*);
- Bloco: é um conjunto de fatos, geralmente em um número fixo predefinido;
- Cadeia de blocos (*blockchain*): conjunto de blocos encadeados (conectados um a um) seguindo uma lógica matemática, ou seja, não são independentes.

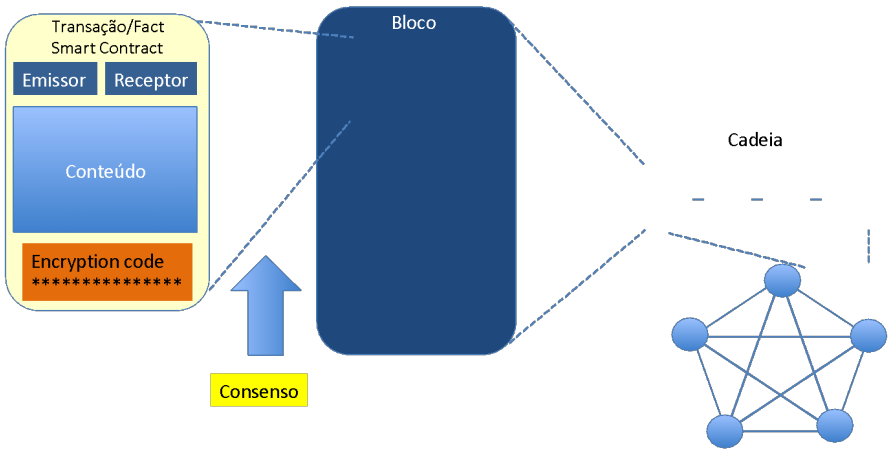
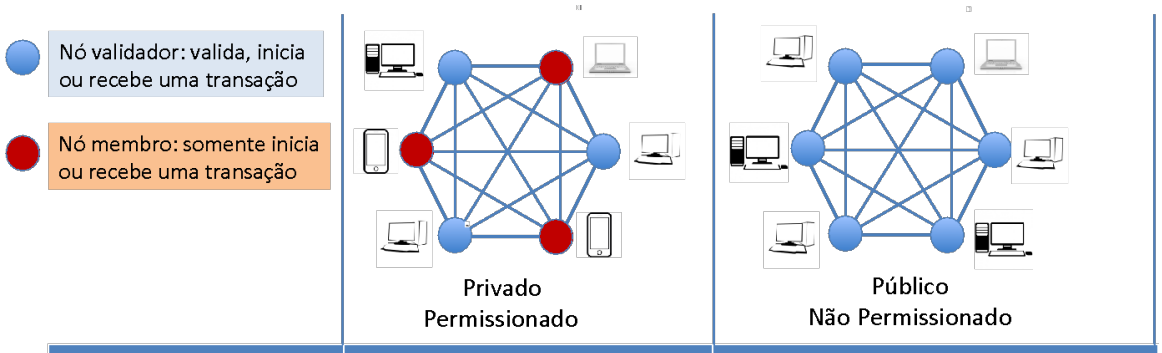


Figura 1. Fato, bloco e cadeia de blocos. Adaptado de [8].

Do ponto de vista de aplicação, a tecnologia *blockchain* passou por uma grande evolução com a possibilidade de uso dos contratos inteligentes que são programas de computador replicados e executados por todos os nós da rede, ou por um conjunto predeterminado de nós denominados validadores. Aplicações baseadas em contratos inteligentes são chamadas *Decentralized Applications* ou Dapps.

Atualmente, as redes *blockchain* são divididas em dois grandes grupos Figura 2:

- ***Blockchains* públicas**: Os protocolos de *blockchain* públicos de última geração são baseados em algoritmos de consenso conhecido como *proof of work* (PoW) são de código aberto e acesso aberto (*permissionless*), qualquer um pode participar sem permissão.
    - Qualquer um pode baixar o código e começar a executar um nó público em seu dispositivo local, validando transações na rede, participando do processo de consenso.
    - Qualquer pessoa no mundo pode enviar transações através da rede e esperar vê-las incluídas no *blockchain* se elas forem válidas.
    - Qualquer pessoa pode ler transações no explorador público de blocos. As transações são transparentes, porém são anônimas ou pseudoanônimas.
- ***Blockchains* privadas**: São as redes de acesso autorizado (*permissioned*). As permissões de gravação são mantidas centralizadas em uma organização. Permissões de leitura podem ser públicas ou restritas a uma extensão arbitrária. Exemplos de aplicativos incluem gerenciamento de banco de dados, auditoria, etc., que são internos a uma única empresa e, portanto, a legibilidade pública pode, em muitos casos, não ser necessária. Os *blockchains* privados são uma maneira de aproveitar a tecnologia *blockchain*, configurando grupos e participantes que podem verificar as transações internamente, essa estratégia pode criar alguns riscos de violação, pois o sistema é centralizado. No entanto, *blockchains* privados têm inúmeros casos de uso, especialmente quando se trata de escalabilidade, conformidade do estado das regras, e privacidade de dados, além de outros problemas de regulamentação.



Acesso à rede	Necessita de autorização	Acesso aberto
Aspectos legais e regulatórios	Alinhado	Não considera, tem regras próprias
Quem são os nós validadores	Grupo pré selecionado	Anônimos
Potenciais aplicações	Ambientes corporativos fechados	Aplicações de acesso aberto (Bitcoin, por exemplo)

Figura 2. Redes *blockchain* privadas e públicas.

Uma outra visão interessante sobre a diferença entre esse dois grandes grupos, pode ser visto na Figura 3 o qual inclui o conceito de federação ou consórcio, *blockchains* federados operam sob a liderança de um grupo. Ao contrário dos *blockchains* públicos, eles não permitem que qualquer pessoa participe do processo de verificação de transações. Os *blockchains* federados são mais rápidos (maior escalabilidade) e fornecem mais privacidade de transações. O nome consórcio normalmente são usados no setor bancário. O processo de consenso é controlado por um conjunto pré-selecionado de nós, por exemplo, pode-se imaginar um consórcio de 15 instituições financeiras, cada uma operando um nó e das quais 10 devem assinar cada bloco para que o bloco seja válido. O direito de ler o *blockchain* pode ser público ou restrito aos participantes.

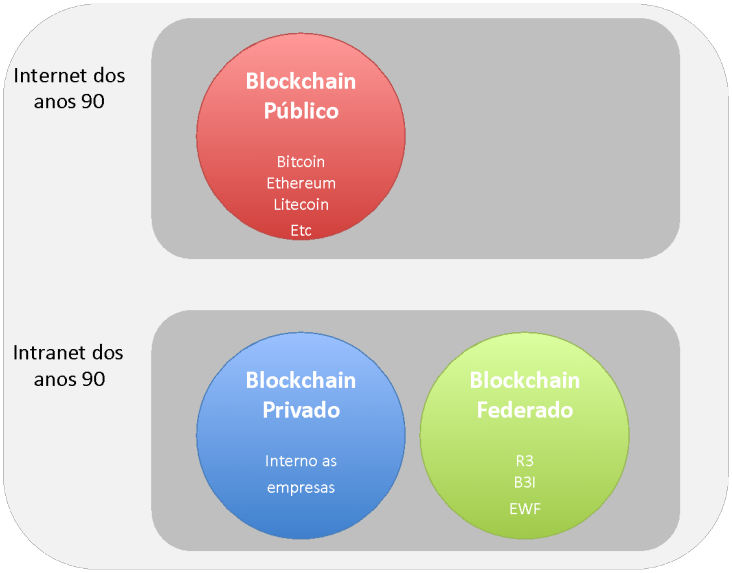


Figura 3. Redes *blockchain* privadas, públicas e federadas. Adaptado de [9].

1.3 Tecnologia *Blockchain* e a Segurança

Apesar dos problemas de segurança, divulgados, utilizando a tecnologia *blockchain*, seja na operação de criptomoedas ou em iniciativas como da DAO [10][11], vale ressaltar que os ataques foram direcionadas as aplicações que utilizam o *blockchain*, e não especificamente a tecnologia ou ainda ao algoritmo empregado.

Além disso, observa-se que os ataques bem-sucedidos, relatados até o momento, a plataformas baseadas em *blockchain*, como por exemplo Bitcoin [12], ocorreram devido a vulnerabilidades nas aplicações e não no core da tecnologia *blockchain* propriamente dito. Sendo assim, com relação ao aspecto de segurança, até o presente momento, não são conhecidas vulnerabilidades contra a construção empregada e algoritmos utilizados nativamente no *blockchain*, podendo assim dizer que segurança ainda é um dos pontos fortes da solução.

Para que isso ocorra o algoritmo prevê que no processo de inserção de novos blocos, um novo bloco, composto por um conjunto de transações, é ligado criptograficamente aos blocos anteriores por meio de um processo chamado de validação. Que nos casos específicos de criptomoedas, Bitcoin por exemplo, é conhecido como mineração. O processo é computacionalmente intensivo e é o que faz com que seja improvável que modificações maliciosas possam ser realizadas por um atacante.

1.3.1 Camadas de Segurança de uma Plataforma *Blockchain*

A plataforma *blockchain* e as aplicações construídas com ele devem adotar a segurança em camadas. Há seis camadas de segurança a serem consideradas. Estas camadas são o resultado da compilação de boas práticas presentes na área de segurança da informação e são apresentadas na Figura 4 e são descritas a seguir [13].

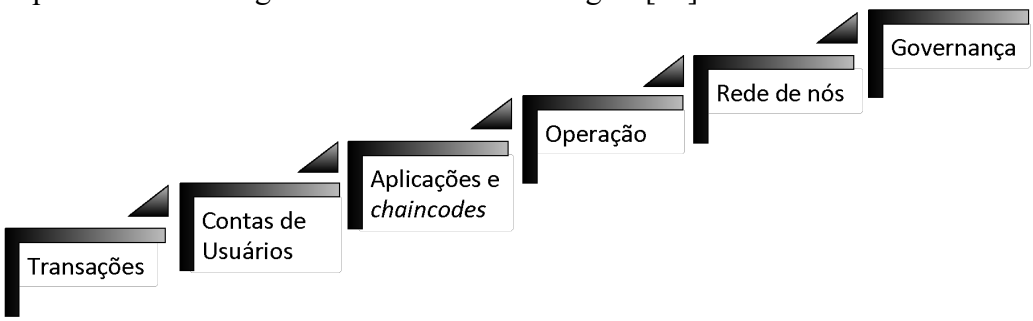


Figura 4. Camadas de segurança para um desenvolvimento *blockchain*.

- A camada fundamental e a primeira camada a ser considerada é a **segurança da transação**. Requisito mínimo sem o qual o *blockchain* não faz sentido. O *blockchain* deve validar as transações com confiança e previsibilidade ao final do consenso. O consenso vai confirmar a finalidade e a imutabilidade de transação.
  Trata-se de proteções sintática e estrutural para as transações e os blocos que as contém. Estas proteções não impedem fraudes semânticas associadas à lógica da aplicação;
- A segunda camada oferece **segurança da conta de usuário**. A conta do usuário é geralmente gerenciada pelo próprio usuário em aplicativos de uso pessoal (*eWallets*). Muitas vezes, a proteção da conta do usuário é confundida com a segurança do software cliente.

Esta camada de segurança é influenciada por dois fatores: a conscientização dos usuários no uso seguro da tecnologia, e a implementação correta dos mecanismos de segurança para dispositivos móveis e sistemas web.

- A terceira camada contempla a **segurança da aplicação e dos *chaincodes***. Fazem parte desta camada as boas práticas de desenvolvimento seguro de software, incluindo a codificação segura de *smart contracts* e a definição de requisitos de segurança, avaliação de arquitetura e testes de segurança da aplicação.
- A quarta camada atende a **segurança de implantação e de operação da aplicação**. Fazem parte desta camada os testes de aceitação e homologação da aplicação e dos *chaincodes* antes de implantação em produção. Uma vez no ambiente de produção, a aplicação deve ser monitorada para detecção de anomalias de funcionamento e comportamento. Monitoramentos avançados podem até detectar fraudes.
- A quinta camada cobre a **segurança da rede P2P de seus nós**. Nesta camada, os mecanismos de proteção tradicionais das redes de computadores (tais como sistemas de *firewall*, IDS, IPS, etc.) podem ser aplicados para proteção dos nós da rede P2P do *blockchain*. Além disso, proteções específicas devem ser aplicadas para a segurança do protocolo de comunicação e de consenso. Ainda, deve ser observada a quantidade mínima necessária de nós disponíveis para garantir o consenso.
- A sexta camada de segurança se refere à **governança da aplicação e do *blockchain***. Esta camada abriga aquelas decisões sobre a estrutura e projeto do *blockchain*, que afetam o funcionamento com segurança, incluindo ainda controles antifraude, auditoria, privacidade e até conformidade a normas padrões específicos do nicho de aplicação.

1.4 *Blockchain* e IoT

Temos que um dos problemas associado à Internet das Coisas (*Internet of Things* – IoT) são os aspectos relacionados a segurança e privacidade, fato esse que alcançou um novo patamar quando a rede de *bots* Mirai causou uma interrupção maciça na Internet em setembro de 2016 [14]. A vulnerabilidade explorada pelo Mirai foi no uso de um ataque baseado em senhas de dicionário em um número, sem precedentes, de dispositivos os quais ofereciam acesso direto à Internet [15][16].

Porém, pesquisas apontam que a utilização da tecnologia *blockchain* pode contribuir para a mitigação deste problema como já discutido em várias publicações dentre elas [17][18][19] onde alguns dos benefícios mais diretos na utilização da tecnologia são:

- Rastrear a história única de cada dispositivo, registrando a troca de dados com outros dispositivos, serviços web e usuários humanos;
- Permitir que dispositivos inteligentes atuem de forma autônoma em uma variedade de transações;
- Monitoramento remoto de ativos de elevado valor para verificar, por exemplo, se estão sendo usados corretamente;

- Monitoramento, controle e autorização de solicitação de determinado equipamento para reposição de alguma peça ou matéria-prima (máquina de lavar solicitando sabão, por exemplo);
- Controle de identidade dos dispositivos IoT para registro e controle de acesso lógico a diferentes aplicações.

Outros exemplos de aplicação envolvendo mais de um setor também podem ser pensados. A tecnologia *blockchain* pode ser aplicada em projetos estruturantes, que envolveriam diferentes atores de uma cadeia de valor, tais como:

- Monitoramento e rastreamento de uma cadeia de produção (p.ex.: fabricação de automóveis, produção de vinhos, produção de equipamentos de informática, dentre outros);
- Sistema de gestão de logística reversa de diferentes produtos (p.ex.: produção de medicamentos, produtos eletroeletrônicos e seus resíduos); e
- Sistemas de gestão e controle da distribuição de venda de produtos sob regime regulatório forte (p.ex.: medicamentos de uso controlado, procedência de carne e alimentos orgânicos).

Em suma, as aplicações para IoT tem potencial para usufruir dos benefícios que a tecnologia *blockchain* traz consigo [20], resumidos na **Tabela 1**.

Tabela 1: Benefícios potenciais da *blockchain* em IoT.

Benefício	Descrição
Transações confiáveis, rápidas e sem intermediário	Reduz ou mesmo elimina o risco da desconfiança entre as partes e custos de transação.
Usuários com poderes	Usuário controla todas as suas transações e informações
Dados de alta qualidade	Os dados da *blockchain* intrinsecamente são completos, consistente, precisos e amplamente disponíveis no momento que forem necessários.
Duráveis, confiáveis e amplamente disponíveis	Ausência de ponto único de falha.
Processos íntegros	Confiança que tudo será executado conforme as regras pré-definidas sem intermediários.
Transparência e imutabilidade	Todas as transações podem estar disponíveis publicamente e não podem ser alteradas ou ainda apagadas dos registros.
Simplificação do ecossistema	Um único livro de registro é criado, reduzindo a desordem e complicações.

À medida que o número de dispositivos conectados cresce de milhões para bilhões, e governos e corporações correm para melhor controlar dispositivos e dados, uma nova estratégia tecnológica será necessária para construir soluções de baixo custo que levem em consideração privacidade e autonomia.

Novos modelos de negócio guiarão estas soluções na direção de economias digitais eficientes e na criação de valor de forma colaborativa, ao mesmo tempo em que serão criados experiência de usuários e produtos melhorados.

No nível mais abstrato, as próprias redes podem se tornar autônomas suplantando sistemas já estabelecidos e que hoje dependem de uma autoridade centralizadora, como, por exemplo, a troca de informação sensível e serviços de auto-instalação e *auto-update* de *software* em dispositivos [21]. Nesse sentido, qualquer

solução IoT descentralizada deveria suportar: (i) mensagens P2P confiáveis, (ii) comunicação intrinsecamente confiável e (iii) autonomia descentralizada.

Em linha com essa perspectiva evolutiva da IoT, uma Cidade Inteligente é um bom exemplo de como combinar IoT e *blockchain*. Os serviços de compartilhamento baseados em *blockchain* podem contribuir para as cidades inteligentes.

A economia compartilhada, nesse caso, é um modelo econômico-social que diversos setores da população podem utilizar para fazer uso compartilhado de ativos subutilizados [22]. Os cidadãos, objetos e *utilities* se conectariam de forma transparente para compartilhar o status e a troca desses ativos.

Nesse paradigma, as pessoas buscam confiança, ter acesso ao invés de ter propriedade, confiabilidade dos serviços compartilhados e segurança e privacidade.

As instituições precisam de uma comunidade inteligente com governo inteligente, parcerias, networking e governança transparentes, interconexão dinâmica com as partes interessadas e estar protegidas de fraudes, responsabilidades e prestadores de serviço desqualificados.

A computação precisa garantir acessibilidade e disponibilidade dos sistemas, recursos de bases de dados inteligentes; sistemas de controle; interface, computação inteligente; habitantes em rede e ciência em tempo real e analíticos avançados.

Há seis elementos que *blockchain* ajudaria nos relacionamentos entre pessoas, tecnologia e organizações: (i) não depender da confiança entre os atores, (ii) transparência e privacidade, (iii) democratização, (iv) automação, (v) ser distribuído e (vi) segurança [23].

1.5 Identidade Digital

As aplicações relacionadas a identidade digital utilizando tecnologia *blockchain* permitem a verificação, autorização e gerenciamento de identidade inalterados, resultando em eficiências significativas e redução de fraudes.

A tecnologia *blockchain* fornece o mecanismo ideal para identidades digitais. Enquanto as identidades digitais estão emergindo como uma parte inevitável do nosso mundo conectado, a forma como protegemos nossas informações on-line está sendo submetida a um intenso escrutínio. Os sistemas de identidade baseados em *blockchain* podem fornecer uma solução para esse problema com criptografia rígida e *ledgers* distribuídos.

Os recentes casos de violações de dados, vazamentos e uso indevido dominaram as manchetes ao longo do ano passado, trazendo nova proeminência às questões de proteção de dados pessoais. O escândalo do Facebook-Cambridge Analytica, a violação de dados do provedor LocationSmart e outros levaram os usuários e os reguladores a examinarem mais de perto como as empresas privadas estão lucrando - e às vezes abusando - dos dados de identidade do cliente. No campo da tecnologia *blockchain* para identidade, vimos as empresas reagirem com um modelo de negócios relativamente novo: o mercado de identidades pessoais [24].

Um pequeno mas crescente contingente de empresas e principalmente *startups* estão desenvolvendo serviços para mudar a monetização de dados pessoais de empresas digitais e anunciantes para os próprios usuários, que são os reais donos da identidade. Esses players dão aos consumidores mais controle sobre como eles “usam” seus dados, combinando funções de identidade e criptomoeda, de forma que os usuários são compensados por atributos individuais que escolhem compartilhar com empresas

privadas. Startups como Datum, DataVest e Wibson, por exemplo, surgiram em 2018 com base nessa funcionalidade. Ainda outras empresas como Civic e Procivis, planejam lançar um novo mercado de *token* totalmente voltado para transações de dados pessoais [24].

Mercados de dados pessoais baseados em *blockchain* são uma proposta intrigante, mas exigem que uma base de consumidores particularmente importante, motivada e digitalmente experiente utilize e consequentemente mude significativamente a economia de dados pessoais.

Até o momento, são poucos os usuários que demonstram interesse em adotar serviços relacionados a privacidade ou o controle adicional sobre suas identidades digitais, portanto os mercados de dados pessoais devem contar com uma estrutura de incentivos forte e experiência de usuário intuitiva para atrair esse público.

1.6 Identidade Digital Autossoberana

Mais de vinte e cinco anos se passaram desde que Peter Steiner mostrou ao mundo pela primeira vez que "Na Internet, ninguém sabe que você é um cachorro" [25], mas esse famoso *cartoon* ainda continua atual e válido, pois representa o desafio de identificar pessoas on-line.

Hoje, estamos muito longe da visão de diretórios públicos, a qual era a expectativa da criptografia de chave pública nos anos setenta ou do grande esquema de certificação hierárquica previsto nos anos oitenta. O gerenciamento de identidades (IdM) na Internet ainda conta com o que Cameron [26], a mais de uma década, chamou de uma "*colcha de retalhos de identidades únicas*", compreendendo vários tipos de sistemas IdM que são restritos a domínios específicos e não interagem entre si.

Os modelos centralizados de IdM enfrentam atualmente inúmeros desafios devido à crescente legislação relacionado às violações de dados, que levam a danos à reputação, fraude de identidade, mas acima de tudo uma perda de privacidade para todos os envolvidos. Esses eventos recorrentes destacam a falta de controle e gestão que os usuários experimentam com suas identidades digitais [27][28][29].

A pesquisa de abordagens alternativas à IdM está sendo conduzida por iniciativas que buscam ampliar a confiabilidade e o alcance das formas digitais de identidade. A Estratégia Nacional dos Estados Unidos para Identidades Confiáveis no Ciberespaço (NSTIC) visa acelerar o desenvolvimento de novas tecnologias que podem aumentar a confiança nas transações on-line [30]. Além disso, o ID2020 procura alavancar tecnologias digitais emergentes para expandir o alcance de identidades legais (espelhando as metas das Nações Unidas de "fornecer até 2030 identidade digital para todos, incluindo o registro de nascimento" [31]). O surgimento do Bitcoin [32] também inspirou um novo pensamento sobre a identidade digital, devido à sua subjacente tecnologia de contabilidade distribuída (DLT), que não precisa de uma autoridade central para validar as transações de sua criptografia natural.

Assim, uma rede globalmente descentralizada é capaz de chegar a um consenso sobre o estado atual das transações. Dado que o DLT é adequado para assegurar o consenso, a transparência e a integridade das transações que ele contém, vários benefícios da aplicação do DLT ao IdM já foram propostos:

- **Descentralizada** - As informações de identidade são referenciadas em um razão que nenhuma autoridade central possui ou controla.

- **Inviolável** - Atividades históricas no DLT não podem ser adulteradas e transparência é dada a todas as mudanças nesses dados.
- **Inclusivo** - Novas maneiras de se criar identidade do usuário podem ser concebidas para expandir o alcance de identidades legais e reduzir a exclusão.
- **Redução de custos** - As informações de identidade compartilhada podem levar à redução de custos para as partes confiáveis, juntamente com o potencial de reduzir o volume de informações pessoais que são replicadas em bancos de dados.
- **Controle de usuário** - Os usuários não podem perder o controle de seus identificadores digitais, mesmo se perderem o acesso aos serviços de um determinado provedor de identidade.

1.7 Identidade Digital Autossoberana Baseado em *Blockchain*

O Gerenciamento de Identidade (IdM) abrange os processos e políticas envolvidos no gerenciamento do ciclo de vida de atributos em identidades para um domínio particular [33].

A maioria dos modelos de IdM hoje é centralizado, onde uma única entidade controla todo o sistema. No entanto, as próprias identidades geradas podem ser federadas além de uma única organização, como quando os governos emitem carteiras de identidade nacionais.

Nos sistemas de identidade federada, os usuários podem usar informações de identidade estabelecidas em um domínio de segurança para acessar outro. Esquemas de *login* único, como o Facebook e Google, por exemplo.

O gerenciamento de identidade centrado no usuário coloca a administração e o controle das informações de identidade diretamente nas mãos dos indivíduos. Os exemplos incluem gerenciadores de senhas (por exemplo, 1Password, Less-Pass, entre outros) que gerenciam, de maneira segura, as diferentes credenciais nos sites da internet.

Apesar das diferentes abordagens, uma função que é fundamental para o IdM é a vinculação segura de um identificador único: um valor que distingue inequivocamente um usuário de outro em um mesmo domínio. E também, atributos (às vezes chamado de certificações ou declarações): direitos ou propriedades de um usuário como nome, idade, classificação de crédito etc.

Os primeiros passos tomados para adequar o uso de DLT para estabelecer um mapeamento de identificador seguro e descentralizado foram tomados no design do *Namecoin* que é um dos *fork* mais longínquos do Bitcoin. O *Namecoin* fornece um *namespace* legível, descentralizado e seguro para o domínio ".bit". Essa conquista contradizia a sabedoria convencional de que um sistema de nomenclatura exibindo todas as três características não poderia ser projetado [34]. *Blockstack* [29] ampliou o esquema do *Namecoin*, para criar uma infraestrutura de chave pública (PKI) descentralizada que registra ligações entre uma chave pública e um identificador legível.

Recentemente, surgiram vários modelos de identidade descentralizada que se estendem além da nomenclatura e visam fornecer um conjunto mais completo de funções de IdM. No entanto, até o momento, não houve uma avaliação direta e ampla dessas propostas.

Atualmente existem basicamente duas categorias de propostas de IdM baseado em *blockchain*:

1. ***Self-Sovereign Identity* (SSI):** uma identidade que pertence e é controlada por seu proprietário sem a necessidade de depender de qualquer autoridade administrativa externa e sem a possibilidade de que essa identidade possa ser removida. Ele pode ser ativado por um ecossistema de identidade descentralizado que facilita o registro e a troca de atributos de identidade e a propagação da confiança entre as entidades participantes. Exemplos incluem Sovrin, uPort e OneName;
2. **Identidade confiável descentralizada:** uma identidade fornecida por um serviço centralizado que realiza a prova de identidade de usuários com base em credenciais confiáveis existentes (por exemplo, passaporte) e registra atestados de identidade em um DLT para validação posterior por terceiros. Exemplos incluem ShoCard, BitID, ID.me e IDchainZ.

1.8 Iniciativas Globais em Identidade Digital Autossoberana

Como exemplo, será apresentado duas iniciativas globais baseadas em SSI, sendo as inciativas da Sovrin e da uPort, e também uma inciativa baseada em identidade confiável decentralizada, nesse caso o da ShoCard.

Acredita-se que com a apresentação dessas três iniciativas globais é possível identificar o modelo de atuação, decisões de design predominantes e também os desafios encontrados. Interessante notar que esses três exemplos servem a um propósito similar para o cenário mais amplo do IdM baseado em DLT.

Essas iniciativas foram escolhidas pois fornecem de forma clara os detalhes técnicos de seus projetos, além disso, são iniciativas sustentadas por comunidades de grande porte ou ainda têm notável nível de financiamento.

1.8.1 Leis da Identidade

Na seção de análise de cada exemplo (Seção 1.8.2.2), (Seção 1.8.3.2), e (Seção 1.8.4.2), serão brevemente analisados perante as conhecidas “Leis da Identidade” [26] que usualmente servem como parâmetros para identificar os sucessos e fracassos dos sistemas de identidade digital. Essas leis compõe uma estrutura conhecida e representa um espectro completo de preocupações com IdM, abrangendo segurança, privacidade e experiência do usuário, as 7 Leis da Identidade são descritas a seguir.

Lei 1 - Controle e consentimento do usuário - As informações que identificam o usuário só devem ser reveladas com o consentimento desse usuário.

Lei 2 - Divulgação mínima e para uso restrito - As informações de identidade só devem ser coletadas em uma base de “necessidade de conhecimento” e mantidas em uma base de “necessidade de reter”.

Lei 3 - Partes justificáveis - As informações de identidade só devem ser compartilhadas com partes que tenham direito legítimo de acessar informações de identidade em uma transação.

Lei 4 - Identidade dirigida - O suporte deve ser fornecido para compartilhar informações de identidade publicamente ou de maneira mais discreta.

Lei 5 - Design para um pluralismo de operadores e tecnologia - Uma solução deve permitir a interoperabilidade de diferentes esquemas de identidade e credenciais.

Lei 6 - Integração humana - A experiência do usuário deve ser consistente com as necessidades e expectativas para que os usuários possam entender as implicações de suas interações com o sistema.

Lei 7 - Experiência consistente em contextos - Os usuários devem ser capazes de esperar uma experiência consistente em diferentes contextos de segurança e plataformas de tecnologia.

1.8.2 *Self-Sovereign Identity* – Sovrin

Sovrin [27] é uma rede de identidade descentralizada de código aberto construída sobre a tecnologia DLT (Figura 5), é considerado uma rede pública / permissionada, onde, apenas as instituições confiáveis chamadas de *stweards* ou *writers* que podem ser bancos, universidades, governos, instituições de pesquisa, etc – são os nós que participam do consenso e executam a gravação na *ledger*. Já os nós observadores só possuem atributo de leitura da *ledger* e atuam como intermediários entre o usuário final e a rede.

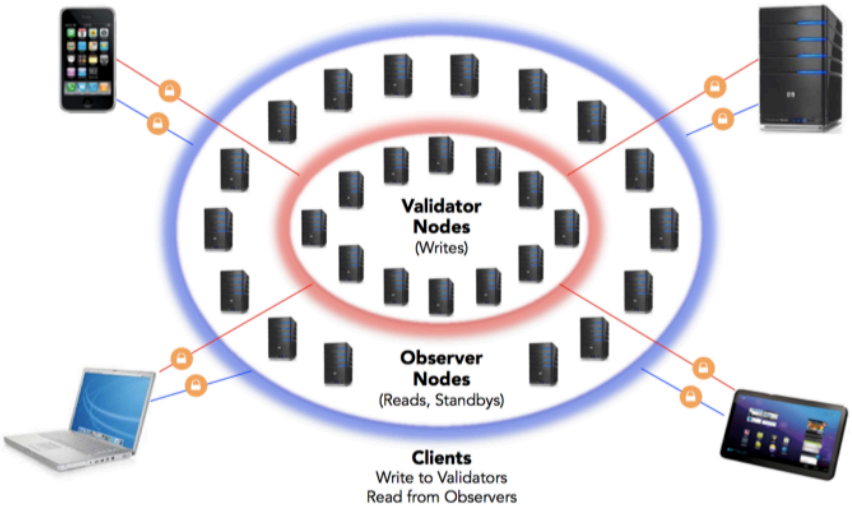


Figura 5. Sovrin: Nós validadores e nós observadores.

A Fundação Sovrin, sem fins lucrativos, assegura a governança adequada dos administradores e o respeito ao acordo legal denominado Sovrin *Trust Framework* firmado entre a fundação e os *stweards/writers*. A Fundação Sovrin é quem fornece o código-base para o projeto Hyperledger Indy [35].

1.8.2.1 Sovrin: Abordagem

A Sovrin permite que um usuário gere tantos identificadores quantos forem necessários para manter a separação contextual de identidades para fins de privacidade. Cada identificador é único, "*unlinkable*" e controlado por um par de chaves assimétricas distintas. Os identificadores da Sovrin são gerenciados pelo usuário ou por um serviço de intermediário designado (agente), e seguem a especificação de Identificador Descentralizado (DID) padrão W3C [36]. Um DID é uma estrutura de dados contendo o identificador do usuário, a chave pública criptográfica e outros meta-dados necessários para transacionar com esse identificador.

A arquitetura da Sovrin pode ser resumida pelos componentes, conforme mostrado na Figura 6. O elemento-chave é a *ledger* Sovrin a qual contém transações

associadas a um identificador específico e é, gravado, distribuído e replicado entre os nós *stewards*, que executam uma versão aprimorada do protocolo redundante de tolerância a falhas bizantina de Aublin et al. [37], chamado Plenum, para consenso [38].

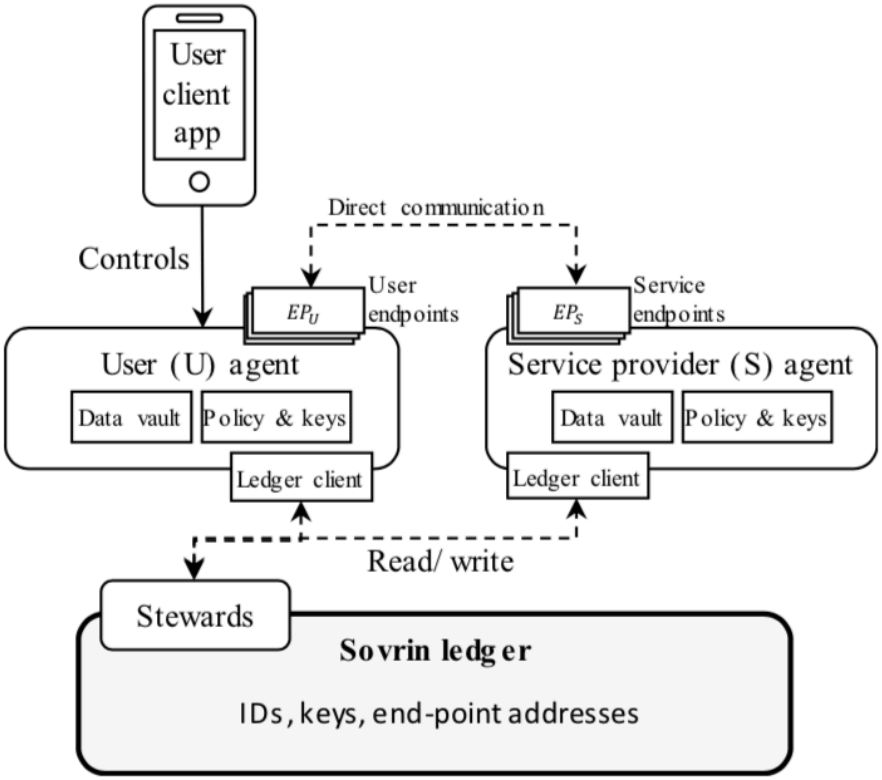


Figura 6. Sovrin: Elementos chave [38].

Há duas consequências importantes para a escolha do consenso ser permissionado no design da Sovrin. Primeiro, não é necessário nenhum cálculo "caro" para chegar a um consenso sobre o estado da *ledger*, reduzindo significativamente o custo de energia da execução de um nó e melhorando drasticamente a performance e tempo de resposta das transações. Em segundo lugar, a confiança na rede Sovrin reside tanto nas pessoas como no código. A confiança começa a partir da raiz comum de confiança formada pelo *ledger* distribuído globalmente, mas à medida que novas organizações e usuários ingressam na rede, eles podem se tornar âncoras de confiança (ou seja, podem adicionar mais usuários e organizações). Espera-se que uma "rede de confiança" evolua para apoiar esse crescimento descentralizado da rede.

Os usuários interagem com a Sovrin através de uma aplicação móvel e controlam os agentes de *software* que agem em seu nome para facilitar as interações com outros agentes na rede (Figura 7). Os *endpoints* de rede devem ser sempre endereçáveis e acessíveis. Os usuários/*endpoints* podem executar as ações em seus próprios servidores, mas provavelmente, a utilização se dará pelos intermediários (agentes) especializados. Os agentes também fornecem serviços de *backup* e armazenamento criptografado das credenciais por exemplo.

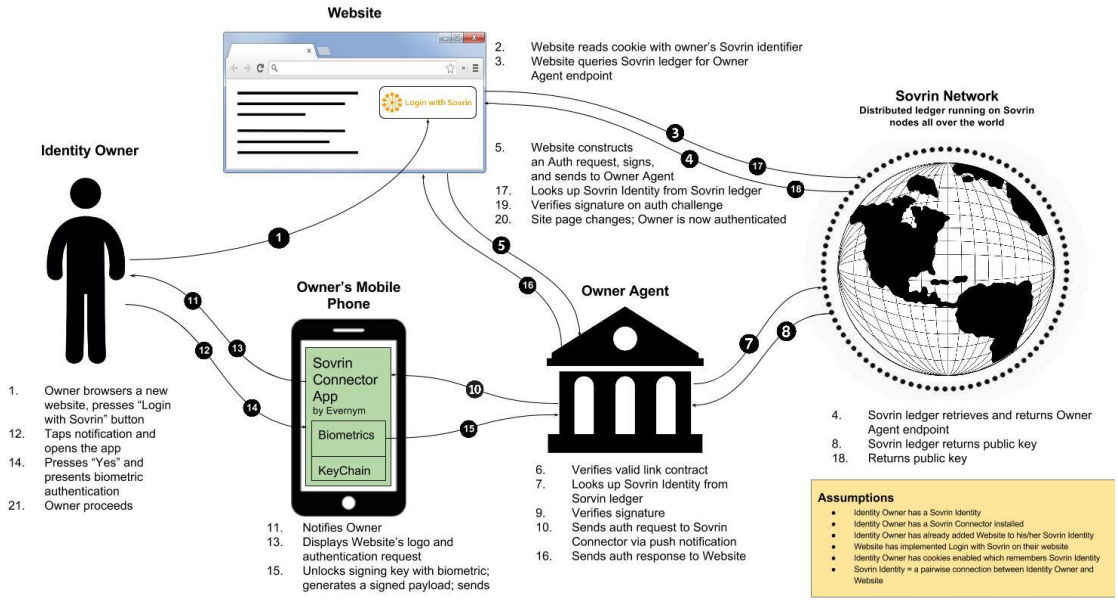


Figura 7. Sovrin arquitetura de comunicação [39].

1.8.2.2 Sovrin: Considerações Baseadas nas Leis da Identidade

A Sovrin visa fornecer aos usuários controle total a todos os aspectos da identidade. Cada usuário pode selecionar as credenciais de atributo que possui sobre si, que deseja compartilhar com uma terceira parte confiável **(Lei 1)**. Isso é possível através do uso de credenciais anônimas.

Embora os usuários possam optar por armazenar esses atributos na *ledger*, em geral, é recomendado que utilizem os recursos de armazenamento dos dispositivos móveis ou ainda recursos dos agentes para transmitir atributos a outras partes por meio de canais de comunicação seguros e usar a *ledger* para identificar a rede e *endpoint* a ser utilizado. O uso de credenciais baseadas em atributos permite que os usuários revelem apenas as credenciais que escolham **(Lei 2)**. A verificação da parte com quem os dados são partilhados continua a ser um desafio, que é parcialmente resolvido através da *web-of-trust*, da governança da Fundação Sovrin e da reputação dos *stewards*.

Embora não haja terceiros de confiança no sentido de PKI na Sovrin, os utilizadores devem confiar nas agências que "agem" em seu nome na rede da Sovrin e nos administradores responsáveis pela manutenção do *ledger*. Dependendo da escolha do agente e de sua implementação, muitas informações podem estar nas mãos desses agentes. No entanto, como as agentes agem em nome do usuário, eles têm um "lugar necessário e justificável" no relacionamento de identidade **(Lei 3)**.

A Sovrin suporta identificadores omnidirecionais e unidirecionais (Lei 4), pois as organizações públicas podem decidir publicar a sua identidade completa na rede, enquanto os utilizadores podem optar por publicar apenas identificadores e usar identificadores e pares de chaves criptográficas diferentes com cada parte com a qual interagem, evitando emitir "identificadores de correlação".

Atualmente a Sovrin ainda depende de um número reduzido de operadores distribuídos geograficamente. À medida que o sistema ganhar notoriedade, com certeza, novos agentes e novos *stewards* irão amentar essa capilaridade. A Fundação Sovrin espera, em particular, construir um mercado de agentes que competirão com as

características que oferecem, por exemplo, mecanismos de backup, carteiras, interfaces com outros sistemas de identidade existentes entre outros **(Lei 5)**.

Finalmente, uma questão importante ainda não amplamente abordada nem pela literatura nem pelos desenvolvedores da rede Sovrin é a experiência do usuário. O histórico de serviços de segurança oferece vários exemplos de sistemas criptográficos inteligentes, que nunca foram implantados amplamente porque os usuários acharam muito complicado ou difícil de entender - a criptografia de e-mail usando o PGP é um exemplo. Portanto, a integração humana continua sendo uma questão em aberto para a rede Sovrin. Considerando que apesar de todos esforço dedicado a arquitetura de sistema e do modelo de SSI empregado na Sovrin, a experiência do usuário não foi amplamente desenvolvido o que penaliza as Leis 6 **(Lei 6)** e 7 **(Lei 7)**.

1.8.3 Self-Sovereign Identity – uPort

O uPort [28] é uma estrutura de identidade descentralizada de código aberto que visa fornecer "identidade descentralizada para todos". Seu caso de uso é o IdM para aplicativos descentralizados de próxima geração (DApps) no Ethereum DLT [40] e para aplicativos tradicionais centralizados, como e-mail e bancos.

1.8.3.1 uPort Abordagem

Uma identidade da uPort é sustentada pelas interações entre os contratos inteligentes da Ethereum. Contratos inteligentes são endereçados exclusivamente por identificadores hexadecimais de 160 bits e, quando invocados, são executados pela Máquina Virtual Ethereum (EVM) instalada em cada nó Ethereum.

Dois modelos de contrato inteligentes projetados pela uPort compreendem cada identidade da uPort: controlador e proxy. Para criar uma nova identidade, o aplicativo móvel uPort de um usuário cria um novo par de chaves assimétricas e envia uma transação para a Ethereum que cria uma instanciação de um controlador que contém uma referência à chave pública recém-criada. Em seguida, é criado um novo proxy que contém uma referência ao endereço do contrato do controlador recém-criado - somente o contrato do controlador pode invocar funções do proxy - uma restrição especificada no controlador e imposta pelo EVM. A Figura 8 fornece uma visão geral de uma interação entre um contrato inteligente e um aplicativo descentralizado no Ethereum.

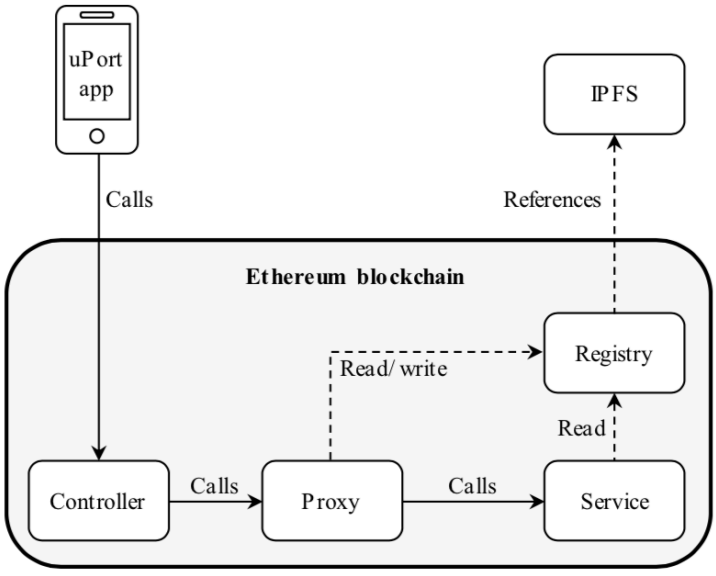


Figura 8. uPort elementos chave [38].

A chave privada que controla um uPortID é armazenada apenas no dispositivo móvel do usuário.

Portanto, um aspecto importante do uPort refere-se ao protocolo de recuperação de ID para o caso de perda ou roubo do dispositivo móvel do usuário. Para isso, os usuários devem indicar nominalmente uPortIDs que terão poder de voto que são os *trustees* (relação de confiança) no processo de substituição da chave pública do usuário solicitante. Um usuário é livre para criar vários uPortIDs que não são vinculados entre IDs.

Um aspecto final do esquema da uPort é o seu suporte para mapear com segurança os atributos de identidade para uma determinada uPortID. O registro da uPort é um contrato inteligente que armazena o mapeamento global de uPortIDs para atributos de identidade. Qualquer entidade pode consultar o registro, no entanto, apenas o proprietário de um uPortID específico pode modificar seus respectivos atributos.

Devido à ineficiência de armazenar grandes volumes de dados em um contrato inteligente, apenas o *hash* da estrutura do atributo JSON é armazenado no registro. Os dados em si são armazenados no IPFS: um sistema de arquivos distribuído em que um arquivo pode ser recuperado por seu *hash* criptográfico.

1.8.3.2 uPort Considerações Baseadas nas Leis da Identidade

A uPort não possui um servidor central e não autentica o proprietário de um uPortID, com isso, o risco de acesso não autorizado são transferidos para os métodos de autenticação local do dispositivo móvel do usuário. Embora o protocolo de recuperação social forneça um método para recuperar a propriedade de um uPortID perdido ou comprometido, os próprios *trustees* podem ser um vetor de ataque. A transparência fornecida a esses curadores, oferece a oportunidades de conluio contra um usuário específico da uPort.

Outro ponto é se um invasor puder comprometer um aplicativo da uPort e substituir os *trustes*, por meio do controlador, o uPortID ficará comprometido permanentemente. Assim, enquanto o uPort coloca mais controle sobre o uPortIDs nas mãos de seus usuários - uma vantagem para **(Lei 1)** - uma camada de complexidade e responsabilidade adicionais é inevitavelmente entregue aos usuários.

A uPort não requer divulgação de dados pessoais para iniciar um uPortID para um uso restrito e também respeita a privacidade em termos da falta de vinculação inerente entre os uPortIDs **(Lei 2)**. No entanto, o registro (quando usado) representa um ponto de centralização que pode ser investigado para obter informações sobre identificadores e dados de identidade. Portanto, embora os atributos específicos na estrutura de dados do atributo possam ser criptografados individualmente, a estrutura de dados JSON geral ainda é visível, o que poderia vazar metadados sobre atributos específicos ou relacionamentos com provedores de identidade/partes confiáveis/ *trustees*. Assim, há uma chance de que o excesso de confiança no registro possa comprometer a privacidade **(Lei 3)**.

Um aplicativo de comércio pode divulgar amplamente seu uPortID, mas o uPort não oferece nenhum diretório público para procurar os uPortIDs a partir de critérios de pesquisa arbitrários. A divulgação discreta de um uPortID é possível se um usuário criar novas uPortIDs para cada nova parte confiável que encontrar e evitar o uso do registro **(Lei 4)**. Embora um uPortID equivale a um contrato inteligente, um nó Ethereum honesto, mas curioso, poderia descobrir até mesmo URLs não divulgados através da análise do código de contrato inteligente armazenado em um determinado endereço para

determinar se é um modelo da uPort, isso implica em um grande trabalho e testes para descobrir se as uPortIDs não divulgadas são privadas na prática, mas com tempo e dinheiro tudo é possível.

A uPort não realiza nenhuma verificação de identidade, mas fornece uma estrutura para os usuários coletarem atributos de um ecossistema de provedores de confiança, mas simplesmente especifica o formato dos atributos armazenados em seu registro. Como consequência do proprietário da uPortID ter acesso de gravação à sua própria parte do registro, um usuário pode descartar seletivamente os atributos negativos que eles recebem, por exemplo: uma contagem de crédito negativa, por exemplo. **(Lei 5)**.

O aplicativo da uPortID fornece uma experiência de usuário consistente em todos os contextos de uso **(Lei 7)** devido à simples leitura de um código QR para iniciar interações com a outra parte confiável. Além disso, não é possível colocar representações de informações pessoalmente identificáveis na DLT o que prioriza a imutabilidade e a transparência dos dados **(Lei 6)**.

1.8.4 Identidade confiável descentralizada – ShoCard

ShoCard [41] fornece uma identidade confiável que aproveita o DLT para vincular um identificador de usuário, uma credencial confiável existente (por exemplo, passaporte, carteira de habilitação) e atributos de identidade adicionais, por meio de *hashes* criptográficos armazenados em transações Bitcoin. Os principais casos de uso da ShoCard são a verificação de identidade em interações presenciais e on-line.

1.8.4.1 uPort Abordagem

A ShoCard usa o plataforma Bitcoin como um serviço de registro de data e hora para *hashes* criptográficos e assinados das informações de identidade do usuário, que são extraídas do Bitcoin. O ShoCard incorpora um servidor central como parte essencial de seu esquema. Esse servidor é intermediário na troca de informações de identidade criptografada entre um usuário e a parte confiável. O esquema se baseia em três fases: (i) bootstrapping, (ii) certificação e (iii) validação. A Figura 9 apresenta essas fases.

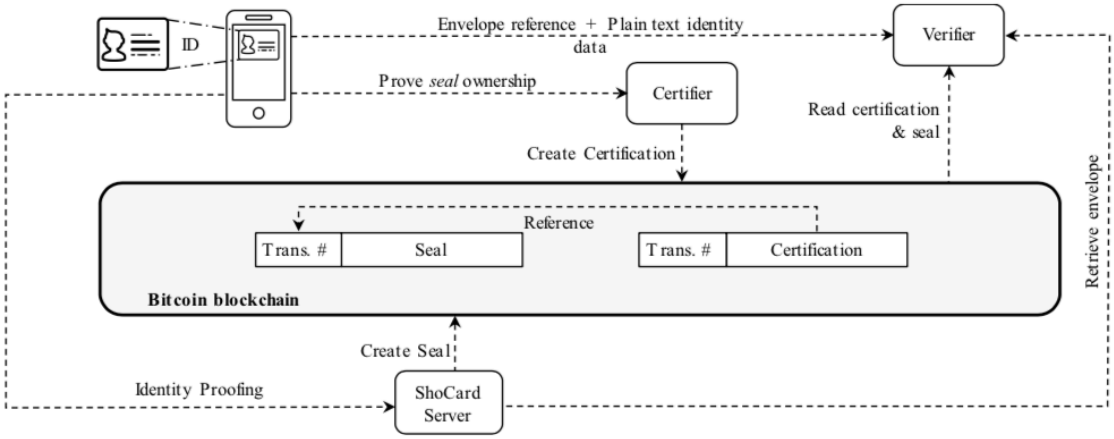


Figura 9. ShoCard elementos chave [38].

Bootstrapping ocorre na criação de um novo ShoCard. O aplicativo móvel ShoCard gera um novo par de chaves assimétricas para o usuário e escaneia suas credenciais de identidade usando a câmera do dispositivo. A varredura e os dados correspondentes são criptografados e armazenados no dispositivo móvel. O *hash*

assinado desses dados também é incorporado em uma transação Bitcoin para fins posteriores de validação de dados. O número de transação resultante do Bitcoin constitui o ShoCardID do usuário e é retido no aplicativo móvel como um ponteiro para o selo ShoCard.

Uma vez que um ShoCard é inicializado, o usuário pode interagir com provedores de identidade para reunir atributos adicionais em um processo chamado certificação. Para associar certificados a um ShoCardID, um provedor de identidade deve primeiro verificar se o usuário conhece os dados com *hash* para criá-lo e as chaves criptográficas que assinaram o selo. Em um contexto face-a-face, isso pode ser alcançado pelo usuário fornecendo os dados de identidade originais que formam o selo de seu dispositivo móvel, um desafio assinado digitalmente e apresentando a credencial confiável original. O certificado assume a forma de um *hash* assinado de novos atributos (e seu associado ShoCardID) em uma transação de Bitcoin criada pelo provedor. O provedor deve compartilhar o número da transação Bitcoin junto com um texto simples assinado dos novos atributos diretamente com o usuário. Como o usuário precisará posteriormente fornecer os atributos para as partes confiáveis e não deseja perdê-los se o dispositivo móvel for perdido, um servidor ShoCard oferece armazenamento para uma versão criptografada de certificações (conhecida como envelope). ShoCard nunca aprende a chave de criptografia, que permite ao usuário compartilhar certificações apenas com partes selecionadas.

A fase de validação ocorre quando uma parte confiante deve verificar uma certificação para determinar se um usuário tem o direito de acessar um serviço (por exemplo, fez o check-in em um voo). Para validar o envelope, o usuário deve primeiro fornecer à parte confiável a referência do envelope e sua chave de criptografia. Depois de recuperar o envelope dos servidores ShoCard, a parte confiante executa várias verificações: i) que a assinatura do envelope foi produzida com a mesma chave privada que assinou o selo; ii) que a assinatura da certificação foi criada por uma entidade de confiança e a certificação em texto simples corresponde àquela com *hash* e assinada na certificação; iii) finalmente, que os dados de identidade apresentados pelo usuário na transação pendente correspondem aos dados assinados e *hash* no selo.

1.8.4.2 ShoCard Considerações Baseadas nas Leis da Identidade

O servidor central ShoCard funciona como um intermediário para gerenciar a distribuição de certificações criptografadas entre os usuários do ShoCard e as partes confiáveis. Desta forma, o Sho-Card tem menos risco de violação de dados do que se armazenasse e distribuísse dados de identidade de texto simples. O armazenamento seguro de informações de identidade e o compartilhamento apropriado com terceiros confiáveis são controlados pelo usuário final **(Lei 1)**. No entanto, o papel intermediário de ShoCard cria incerteza sobre a existência longitudinal de um ShoCardID. Se a empresa deixasse de existir, os usuários do ShoCard não poderiam usar o sistema com as certificações adquiridas. Isso torna o ShoCard mais centralizado na prática do que sua confiança aberta no DLT poderia sugerir.

Cada identidade de ShoCard é inicializada com uma credencial confiável existente, como um passaporte ou carteira de motorista. Tal abordagem pode exigir que os usuários incorporem mais informações pessoais em seu selo ShoCard do que pretendiam originalmente. Isto pode tornar o ShoCard menos atraente para contas online de baixo valor **(Lei 2)**.

Como o usuário está no controle de iniciar atividades de compartilhamento e como o ShoCard armazena apenas dados criptografados, pode haver alguma confiança de que apenas partes justificáveis estão envolvidas na transação de compartilhamento de dados de identidade. No entanto, o servidor ShoCard pode ser capaz de associar um determinado ShoCardID a uma terceira parte confiável, já que os envelopes devem ser recuperados do servidor ShoCard pela terceira parte confiável **(Lei 3)**.

ShoCard suporta apenas identificadores unidirecionais e não possui o conceito de registro público de ShoCardIDs. Embora identificadores omnidirecionais possam ser necessários no futuro para realizar sua visão de um ecossistema de certificações reutilizáveis **(Lei 4)**.

A ShoCard suporta uma infinidade de provedores de identidades diferentes por meio de sua funcionalidade de certificação, mas esses provedores devem criar uma integração personalizada com os próprios serviços web da Sho-Card, além do Bitcoin, o que poderia ser uma barreira à aceitação. A decisão de realizar essa integração pode ser motivada pela confiabilidade da comprovação de identidade de seus usuários pelo Sho-Card **(Lei 5)**.

A digitalização de documentos de identidade e baseado em códigos QR o que é um paradigma de interação dominante na experiência do usuário do ShoCard - é simples e consistente **(Lei 7)**. No entanto, não está claro quais seriam as motivações do usuário para adotar esse novo tipo de identidade digital, e como os usuários seriam educados sobre as implicações da referência de dados de identidade em rede *blockchain* **(Lei 6)**. Os usuários também não são compatíveis com o gerenciamento de chaves criptográficas.

Um último ponto diz respeito à disponibilidade geral do ShoCard. As transações de Bitcoin levam em média 10 minutos para serem exploradas na rede *blockchain* e, além disso, é recomendável aguardar que seis blocos adicionais sejam extraídos antes de assumir a liquidação de uma transação. Isso poderia levar em média o tempo de espera para a liquidação a uma hora. Se um contexto depender da liquidação em tempo real das certificações, essa prazo poderá criar desafios com relação a experiência do usuário e a adoção pelos fornecedores.

1.9 Aspectos legais e Padronização da Identidade Digital Autossoberana

a) Aspectos Legais

Um dos grandes desafios da identidade digitais autossoberana é o atendimento dos requisitos das principais leis relacionadas com a proteção de dados pessoais. Vários países adotaram um modelo jurídico para proteção de dados pessoais através de um regime legal de proteção de dados, na forma de uma lei geral. Com exceção dos Estados Unidos, a maioria dos países desenvolvidos, e também o Brasil, aprovaram leis abrangentes contemplando os setores públicos e privado. Embora alguns países possuam suas leis gerais, estas podem coexistir com normas setoriais, regulando setores específicos de forma complementar às leis gerais [42].

Dentre estas leis, destaca-se o Regulamento Geral de Proteção de Dados da União Europeia (RGPD), também conhecido como *General Data Protection Regulation* (GDPR), que foi elaborado pelo Parlamento Europeu e Conselho da União Europeia e publicado no dia 04 de maio de 2016. Ele foi implementado nos 28 países membros da União Europeia em 25 de maio de 2018. Ele se aplica à proteção das pessoas naturais no que diz respeito à proteção de dados e também ao livre movimento desses dados e

revoga a Diretiva de Proteção de Dados Pessoais de 1995 (95/46/CE). O regulamento, que na União Europeia tem força de lei, possui um conteúdo bastante extenso, com 173 considerandas e 99 artigos.

No Brasil, a Lei 13.709/2018, também conhecida como Lei Geral de Proteção de Dados brasileira (LGPD), que foi publicada em 14 de agosto de 2018 e, segundo COTS [43], com esta publicação "o Brasil se integrou, não sem um certo atraso, ao grupo de países que possuem legislações específica para proteção de dados pessoais". Pode-se afirmar que a grande fonte de inspiração para a elaboração da LGPD foi o GDPR, sendo a primeira mais genérica e, consequentemente, menos detalhada que o regulamento.

Seguem algumas questões relevantes das leis gerais de proteção de dados pessoais que podem impactar soluções que utilizam blockchain, com destaque para a identidade digital autossoberana:

- **Direto de apagar ou direito de ser esquecido**: Em algumas algumas situações, o titular tem o direito de obter do responsável pelo tratamento o apagamento dos seus dados pessoais, sem demora injustificada, e este tem a obrigação de apagar os dados pessoais, sem demora injustificada. Portanto, tal direito inviabiliza o registro de dados pessoais na ledger. Nas propostas de identidade autossoberana, os dados pessoais nunca são colocados na ledger. Em vez disso, são colocados somente identificadores pseudônimos e descentralizados denominados *Decentralized Identifiers* (DIDs) [44], chaves públicas pseudônimas, endereços de agentes e as estruturas das credenciais emitidas (schemas), conforme especificado pela W3C [45] . Isso permite que a troca de dados pessoais ocorra inteiramente fora da ledger. Vale destacar que, diferentemente do GDPR, na LGDP não tem previsão específica para tratamento do direito de ser esquecido. Segundo ministro do STJ Paulo de Tarso Sanseverino "A LGPD abrange todos os dados pessoais, inclusive digitais. O Marco Civil tem a preocupação somente com os efeitos da Internet. Apesar disso, a nova legislação não tem previsões importantes, como é o caso do Direito ao esquecimento" [46]
- **Direito de retificação**: O titular tem o direito de obter, sem demora injustificada, do responsável pelo tratamento a retificação dos dados pessoais inexatos que lhe digam respeito. Como os dados pessoais não são colocados na *ledger* e, geralmente, ficam sobre a gestão do titular, este requisito também é atendido pelas soluções de identidade digital autossoberana;
- **Direito de acesso**: Isso significa que os titulares de dados têm o direito de perguntar a um controlador de dados se seus dados pessoais estão sendo processados e, se forem, receber detalhes sobre como este processamento se dá e onde. No caso da identidade digital autossoberana quem controla o acesso aos dados é o próprio titular através dos DIDs;
- **Portabilidade dos dados**: O direito à portabilidade de dados (artigo 20.º do GDPR, por exemplo) permite que um titular de dados receba dados de um responsável pelo tratamento, a fim de os transmitir a outro controlador [47]. O Grupo de Trabalho do Artigo 29[1], por exemplo, considera que o "principal

[1] Trata-se da designação abreviada do Grupo de Proteção de Dados estabelecido pelo artigo 29.º da Diretiva 95/46/CE. Proporciona à Comissão Europeia aconselhamento independente sobre questões de proteção de dados e contribui para o desenvolvimento de políticas harmonizadas de proteção de dados nos Estados-Membros da UE. O grupo de trabalho é composto por todos os representantes das autoridades nacionais de supervisão dos Estados-Membros da UE.

objetivo da portabilidade de dados é aumentar o controle dos indivíduos sobre seus dados pessoais e garantir que eles desempenhem um papel ativo no ecossistema de dados". A maioria das soluções atuais não fornecem aos proprietários tal funcionalidade. Isso não se aplica a identidade digital autossoberana, onde a gestão dos dados (armazenamento e controle de acesso) é definida pelo próprio usuário dos dados, podemos armazenar estes dados nos seus próprios dispositivos através de um *Mobile Edge Agent*[2] ou ainda na nuvem usando um *Hub* que armazena e compartilha dados em nome dos seus proprietários, podendo ser concebido como uma carteira remota, que armazena todos os dados anonimamente criptografados, mas não as chaves.

b) Os esforços de padronização

Atualmente, existem ações globais no sentido de buscar padronização para os diferentes protocolos e agentes que constituem as soluções de identidade digital autossoberana. Conforme apresentado na figura X [48], existem vários órgãos envolvidos nestes esforços de padronização, com destaque para o W3C[3] (*World Wide Web Consortium*) envolvido na padronização dos DIDs e das *Verifiables Credentials*.

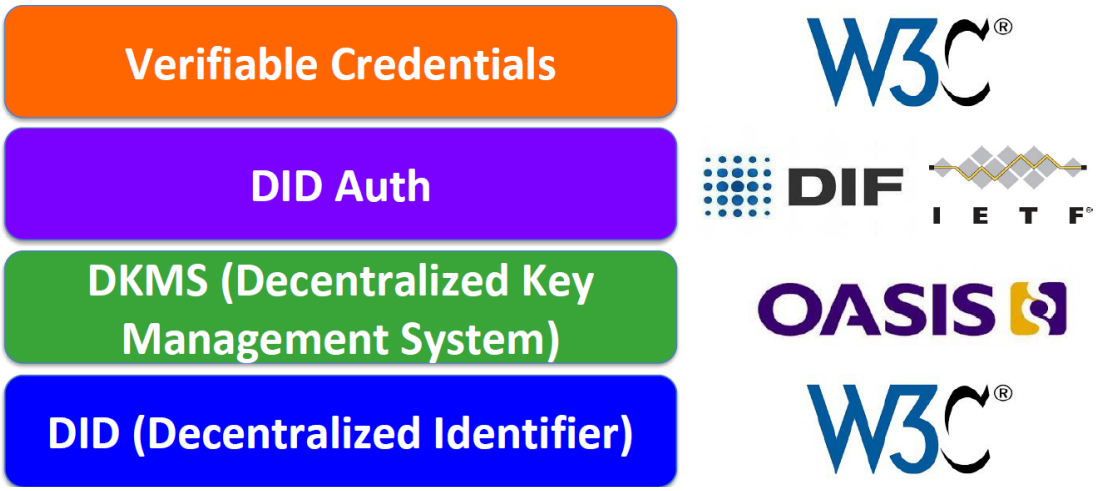


Figura 10 - Esforços de Padronização

Muitos destes padrões ainda estão em fase de discussão. Um exemplo é a própria padronização dos DIDs, com o W3C lançando recentemente a versão V.013 do *Data Model and Syntaxes* [49].

DKMS (*Decentralized Key Management System* ou Sistema de Gerenciamento de Chaves Descentralizadas) é um padrão aberto emergente para gerenciar DIDs e chaves privadas. O DKMS se aplica às carteiras onde se armazenam DIDs e chaves privadas, assim como aos agentes que leem/escrevem nessas carteiras. A ideia do DKMS é padronizar as carteiras para que o usuário nunca precise se preocupar com a segurança, a privacidade ou o bloqueio de fornecedores. A arquitetura inicial do DKMS está agora em análise pública aberta no *Hyperledger Indy github* [50].

[2] Mobile Agents são aplicações com as quais as pessoas interagem diretamente para controlar sua identidade, gerenciando convites para criar Identificadores Descentralizados (DIDs) com outros agentes, gerenciando a concessão de dados pessoais e etc. Tudo isso no dispositivo mobile do dono da identidade.

[3] W3C - O World Wide Web Consortium é a principal organização de padronização da World Wide Web.

O *DID Auth* é uma forma padrão simples para um proprietário de DID autenticar, comprovando o controle de uma chave privada. Em novembro de 2017 foi formado o grupo de trabalho *DID Auth* pela *Decentralized Identity Foundation*[4]. Em fevereiro de 2018, foram lançadas as especificações e implementações preliminares do padrão. Em abril de 2018, foi apresentado o primeiro protótipo do padrão no *Internet Identity Workshop*.

As *Verifiable Credentials* é um formato para credenciais digitais interoperáveis e criptograficamente verificáveis que estão sendo definidas pelo *Verifiable Claims Working Group* do W3C criado em maio de 2017. A missão do grupo é tornar mais fácil e mais segura a divulgação e troca de credenciais que foram verificadas por terceiros [51].

Por enquanto, a discussão sobre padronização de identidade digital autossoberana no Brasil está sendo contemplada na comissão ABNT/CEE-307 - *Blockchain* e Tecnologias de Registro Distribuídas, que possui no seu âmbito de atuação a normalização no campo de *Blockchain* e tecnologias de registro distribuídas, no que concerne a terminologia e generalidades. Esta Comissão é espelho do ISO/TC-307 – *Blockchain* and *Distributed Ledger Technologies.*

Recentemente, a comissão divulgou a versão preliminar do documento "*Blockchain* e tecnologias de registro distribuídas" – Conceitos e elementos da tecnologia *Blockchain*" composto por seis partes. O tema identidade está sendo abordado na sexta parte do documento (segurança, privacidade e identidade).

1.10 A importância e os desafios das Carteiras Digitais (*"Digital Wallets"*)

Carteiras Digitais são peça chave para soluções de Identidade Digital Autossoberana, elas não servem apenas para guardarmos de forma segura e confiável nossas credenciais, chaves privadas e informações, além disso, elas servem também para gerenciarmos para que agentes terceiros possam, com o nosso consentimento, acessar às nossas credenciais e dados, nos dando meios para verificar quem possui tais permissões e também poder para revogar permissões que não queiramos mais ou que não fazem mais sentido. Além disso, Carteiras Digitais são responsáveis por fazer o gerenciamento de nossas credenciais, desde comprovantes de conclusão de cursos, que devem ficar por um longo tempo (pelo menos enquanto forem relevantes) até mesmo a simples ingressos para eventos (por exemplo, cinema e teatro), que poderiam ser descartados (ou movidos para um arquivo morto) apenas a algumas horas depois de seu uso.

Apesar de indubitável que carteiras digitais será amplamente usada para o gerenciamento e controle de nossas identidade e dados, seu conceito possui mais de duas décadas. Um exemplo de uma carteira digital criada no início dos anos dois mil, é o Microsoft Passport [52], que se propunha a ser a solução informações para *"single signon"* e de cartões de créditos em um único lugar. O Microsoft Passport fracassou, sem nunca ter chegado perto da quantidade de usuários e aplicações integradas que um dia se pensou para ele. Assim como o Microsoft Passport, inúmeras outras soluções de Carteiras Digitais fracassaram ao longo dos últimos vinte anos, em geral, os principais motivos de fracassos foram, dentre outros, os seguintes aspectos:

[4] O DIF é uma organização focada no desenvolvimento dos elementos fundamentais necessários para estabelecer um ecossistema aberto para a identidade descentralizada e garantir a interoperabilidade entre todos os participantes.

- As Carteiras Digitais criadas nesse período, em sua maioria, são iniciativas fechadas, não possuem padronização e ou meios para portabilidade, fazendo com que os usuários se tornassem reféns dessas soluções;
- Problemas de segurança, como os reportados para a abordagem de *single signon* [53];
- Essas iniciativas foram ambiciosas demais, propondo-se a gerenciar e a controlar todas as credenciais dos usuários de uma única vez, inevitavelmente se tornando, por tanto, soluções complexas demais e, em geral, enfadonhas de se usar.

Além dos desafios citados acima, que continuam sendo uma realidade a ser enfrentada por Carteiras Digitais modernas, também vale ressaltar outros desafios como, por exemplo [54]:

- Portabilidade entre carteiras (o usuário, tendo controle de sua identidade, deve ser capaz e ter meio para portar e controlar suas credenciais na solução que preferir e mais confiar);
- Padronização de credenciais, sendo que, em tese, o *schema* de uma credencial financeira emitida por um *agent*, deveria ser minimamente compatível com o esquema da credencial de um outro *agent* qualquer do mesmo segmento;
- Facilidade para realizar backup, vez que acidentes acontecem e, perda de dispositivos móveis (onde, eventualmente, as carteiras digitais de pessoas físicas poderiam ficar armazenadas), ou até mesmo em caso de dano de aparelho ou exposição da carteira, em todos esses casos, deveria ser possível ao usuário revogar suas credenciais expostas e conseguir emitir novas e recuperar os dados perdidos;
- E por último, mas tão crucial quanto os demais, é a usabilidade das carteiras digitais. Carteiras digitais confusas, com enormes quantidades de credenciais antigas que não fazem mais sentido e que não são fáceis de usar ou cujas funções não agreguem valor e ou facilidade ao seu portador, são os grandes motivos para que usuários não façam o uso ordinário das novas carteiras digitais, como por exemplo, Google Wallet ou Apple Wallet.

Além dos desafios acima demonstrados, será também necessária a criação de metodologias e, eventualmente, até mesmo de certificação de segurança para carteiras digitais, afins de garantir ante ao portador da credencial que aquela solução que armazenará sua representação digital ante ao mundo (isto é, sua identidade digital) é confiável e segura o bastante para que ele possa usá-la

1.11 Criptografias BLS-Signature e CL-Signature para criação de verfinym e pseudonym

O framework para gerenciamento de Identidade Digital Descentralizada do projeto Hyperledger, o Hyperledger Indy, possui uma biblioteca de criptografia chamada *"indy-crypto"*, que atualmente é usado para "Provas Zero de Conhecimento" (*"ZKPs"*) e para esquemas de assinaturas digitais para credenciais verificáveis (*"Verifiable Credentials"*) emitidas entre os agentes.

Resumidamente, um esquema de assinatura digital possui ao menos as três principais funções a seguir:

- **Geração de um par de chaves**, pública e privada (*"key pair"*);
- Processo de operação para **assinatura digital de uma mensagem** que, em geral (mas, não exclusivamente), é realizado com o uso da chave privada;
- **Verificação da assinatura digital**, usando a chave pública.

O esquema de assinatura adotado pelo framework Hyperledger Indy para permitir verificar se a assinatura é verdadeira, é o Boneh-Lynn-Shacham (BLS) [55].

O esquema BLS usa o conceito de emparelhamento bilinear (*"bilinear paring"*) [56] para as operações de verificações de assinaturas, fazendo o gerenciamento das assinaturas por grupos de curvas elípticas (cada assinatura pertence a um determinado grupo de curva elíptica), tal abordagem de trabalhar com grupos de curvas elípticas provê defesas sólidas e eficientes contra ataques de *Index Calculus* [57], permitindo assim a criação de assinaturas menores que as do padrão FDH, mas com o mesmo nível de segurança [58].

O emprego do esquema BLS permite também, a agregação de múltiplas assinaturas para uma mesma mensagem, criando assim, a possibilidade de realizar uma análise criptográfica a fim de constatar qual o remetente(s) e o destinatário de uma determinada mensagem assinada [59] com esquema BLS, sendo esta, então, a abordagem do conceito de credenciais verificáveis (*"Verifiable Credentials"* e *"Verifinym"*) do Hyperledger Indy.

Além da BLS, outro esquema de assinatura digital encapsulado pela biblioteca indy-crypto é o Camenisch-Lysyanskaya (CL), esquema este usado para a geração de validações ZKPs e a criação de credenciais anonimizadas (*"Pseudonym"*) [60].

Esses dois esquemas de assinaturas digitais, mais o protocolo *Blockchain* específico para o gerenciamento de transações para Identidade Digital Autossoberana, são os pilares do framework Hyperledger-Indy e o surgimento de uma rede para Identidade Digita Autossoberana.

1.12 Requisitos de sistema

Para o correto desenvolvimento de soluções para Hyperledger Indy, o sistema operacional que deve ser utilizado é o Ubuntu 16.04 LTS com a seguinte configuração mínima de hardware:

- Processador Core-i5 terceira geração ou superior;
- 6 Gigabytes de memória RAM ou superior;
- Mínimo de 80 Gigabytes de memória em disco disponível.

1.13 Tutorial Hyperledger Indy

Para preparar o ambiente de desenvolvimento voltado para .Net são necessários alguns pré-requisitos.

Visual Studio Code.

https://code.visualstudio.com/

- Após a sua instalação adicione as extensões C# e *Nuget package* manager

.Net Core na versão 2.2.

Instalando o Indy-SDK.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
68DB5E88
sudo add-apt-repository "deb https://repo.sovrin.org/sdk/deb
xenial stable"

sudo apt-get update
sudo apt-get install -y libindy
```

Se você baixar o repositório libindy da *master* verifique a biblioteca libindy.so está em /usr/lib/libindy.so senão copie para este diretório.

Se preferir há a opção construir o ambiente de desenvolvimento gerando os binários a partir do código fonte. Siga os passos descritos no link [https://github.com/hyperledger/indy-sdk/blob/master/docs/build-guides/ubuntu-build.md]
Instalando o *Docker*:
https://docs.docker.com/

A maneira recomendada de iniciar um nó de rede local (um pool) é utilizando o *Docker*. Execute os seguintes comandos no terminal. Em seguida configure o IP do *Docker* container e do pool de forma que eles correspondam.

```
docker build -f ci/indy-pool.dockerfile -t indy_pool .
docker run -itd -p 9701-9708:9701-9708 indy_pool
```

1.14 Criando um projeto.

Com o ambiente já configurado, iniciaremos o tutorial com operações básicas da biblioteca libindy. Em nosso exemplo vamos criar uma negociação de prova de confiança. Para elucidar, imagine que uma instituição peça informações para você e, para provar, você apresente um documento emitido pelo governo ou órgão de confiança sob a governança da uma federação.

A negociação de prova normalmente começa quando um verificador solicitar uma prova de veracidade de informação. Uma solicitação de prova é um arquivo JSON que descreve que tipo de prova satisfaria a parte confiável. Depois que a solicitação de prova é recebida, um detentor de credenciais deve verificar sua carteira de identidade para descobrir quais credenciais podem ser usadas para atender à solicitação

Para iniciar, crie um projeto no *Visual Code* (neste exemplo um projeto do tipo console *application*):

```
dotnet new console -lang C# -n <nome-do-seu-projeto>
```

Importe o pacote *HyperLedger Indy SDK* e *Newtonsoft* via *nuget* para o projeto
https://www.nuget.org/packages/Hyperledger.Indy.Sdk
https://www.nuget.org/packages/Newtonsoft.Json/

Crie um arquivo na raiz do projeto com o seguinte conteúdo abaixo e prossiga da mesma forma para criar um total de 4 nós de rede ou copie docker_pool_transactions_genesis em "indy-sdk/cli/docker_pool_transactions_genesis" e para mapear o container docker mude o "client_ip" e o "node_ip" para de "10.0.0.2" para "127.0.0.1" para usar localhost.

Configuração do nó 1:

```
{
   "reqSignature": {},
   "txn": {
       "data": {
           "data": {
               "alias": "Node1",
               "blskey":
"4N8aUNHSgjQVgkpm8nhNEfDf6txHznoYREg9kirmJrkivgL4oSEimFF6
nsQ6M41QvhM2Z33nves5vfSn9n1UwNFJBYtWVnHYMATn76vLuL3zU88Ky
eAYcHfsih3He6UHcXDxcaecHVz6jhCYz1P2UZn2bDVruL5wXpehgBfBaL
Km3Ba",
               "blskey_pop":
"RahHYiCvoNCtPTrVtP7nMC5eTYrsUA8WjXbdhNc8debh1agE9bGiJxWB
XYNFbnJXoXhWFMvyqhqhRoq737YQemH5ik9oL7R4NTTCz2LEZhkgLJzB3
QRQqJyBNyv7acbdHrAT8nQ9UkLbaVL9NBpnWXBTw4LEMePaSHEw66RzPN
dAX1",
               "client_ip": "127.0.0.1",
               "client_port": 9702,
               "node_ip": "127.0.0.1",
               "node_port": 9701,
               "services": [
                   "VALIDATOR"
               ]
           },
           "dest":
"Gw6pDLhcBcoQesN72qfotTgFa7cbuqZpkX3Xo6pLhPhv"
       },
       "metadata": {
           "from": "Th7MpTaRZVRYnPiabds81Y"
       },
       "type": "0"
   },
   "txnMetadata": {
       "seqNo": 1,
```



```
       "txnId":
"fea82e10e894419fe2bea7d96296a6d46f50f93f9eeda954ec461b2e
d2950b62"
   },
   "ver": "1"
}
```

Copie as seguintes bibliotecas:

```
using System;
using Hyperledger.Indy.PoolApi;
using Hyperledger.Indy.WalletApi;
using Hyperledger.Indy.LedgerApi;
using Hyperledger.Indy.DidApi;
using Hyperledger.Indy.AnonCredsApi;
using Newtonsoft.Json.Linq;
```

Passo 1: Criando um DID

Nesta primeira parte será criado um identificador descentralizado (DID) e posteriormente será realizada a consulta de informação na *ledger* em função do *Trust Anchor* (nó de confiança). Para isso, será necessário um *pool* de nós do Hyperledger Indy para as transações na *ledger*. Na sequência, será criada uma carteira digital onde para controle manipular uma carteira onde será alocado o DID com a par de chaves e as credenciais do *Trust Anchor* e do Steward. Posteriormente será feito uma requisição para o Steward submeter o DID do *Trust Anchor* (nó de confiança) na *ledger* com a assinatura de ambos com as suas respectivas chave privada.

A título de exemplo usaremos apenas uma carteira, mas os agentes e Steward devem ter suas próprias carteiras para estabelecer uma canal de comunicação em uma situação real.

Na configuração do pool é necessário referenciar o diretório do docker pool. Lembrando que o endereço IP do container docker deve estar devidamente configurado e corresponder ao do pool. Certifique-se de subir o container criado e copie o código abaixo para a classe demo do projeto

Obs: O *seed* do did declarado inicialmente deve conter 32 caracteres e funciona como uma chave de administrador da rede.

```
Console.WriteLine("Selecionando a versão Indy Node 2.0");
       Pool.SetProtocolVersionAsync(2).Wait();

       string poolName = "pooldemo";
       string startupPath = Environment.CurrentDirectory;
       string poolConfig = "{\"genesis_txn\": \"" +
startupPath + "/docker_pool_transactions_genesis\"}";
       string issuerWalletConfig =
"{\"id\":\"issuerwallet\"}";
       string issuerWalletCredencial =
"{\"key\":\"issuerwalletkey\"}";
       string did = "{\"seed\":
\"000000000000000000000000Steward1\"}";
```

```
      Console.WriteLine("# 1  Criando a configuração de
um pool local na ledger que será usado posteriormente
para conectar o pool de nós da rede.");
      Pool.CreatePoolLedgerConfigAsync(poolName,
poolConfig).Wait();

      Console.WriteLine("# 2 Abrindo o pool na ledger e
obtendo seu gerenciador da libindy.");
      Pool poolHandle =
Pool.OpenPoolLedgerAsync(poolName, "{}").Result;

      Console.WriteLine("# 3 Criando uma  identity
wallet.");
      Wallet.CreateWalletAsync(issuerWalletConfig,
issuerWalletCredencial).Wait();

      Console.WriteLine("# 4 Abrindo identity wallet e
obtendo seu gerenciador da libindy.");
      Wallet issueWalletHandle =
Wallet.OpenWalletAsync(issuerWalletConfig,
issuerWalletCredencial).Result;

      Console.WriteLine("# 5 Gerando e armazenando o DID
Steward e a chave-valor(verkey).");
      CreateAndStoreMyDidResult DidSteward =
Did.CreateAndStoreMyDidAsync(issueWalletHandle,
did).Result;
      Console.WriteLine(string.Format("Steward Did:
{0}", DidSteward.Did));
      Console.WriteLine(string.Format("Steward
Verkey:{0}", DidSteward.VerKey));

      Console.WriteLine("# 6 Gerando e armazenado o Did
do Trust Anchor e a verkey");
      CreateAndStoreMyDidResult DidTrustAnchor =
Did.CreateAndStoreMyDidAsync(issueWalletHandle,
"{}").Result;
      Console.WriteLine(string.Format("Trust Anchor DID:
{0}", DidTrustAnchor.Did));
      Console.WriteLine(string.Format("Trust Anchor
Verkey: {0}", DidTrustAnchor.VerKey));

      Console.WriteLine("#7 Construindo uma requisição
Nym para adicionar o Trust Anchor na ledger");
      var nymRequest =
Ledger.BuildNymRequestAsync(DidSteward.Did,
DidTrustAnchor.Did, DidTrustAnchor.VerKey, null,
"TRUST_ANCHOR").Result;
```



```
       Console.WriteLine(string.Format("Nym Request:
{0}", nymRequest));

       Console.WriteLine("#8 Enviando a requisição Nym
para a ledger");
       var nymResponse =
Ledger.SignAndSubmitRequestAsync(poolHandle,
issueWalletHandle, DidSteward.Did, nymRequest).Result;
       Console.WriteLine(string.Format("Nym Response:
{0}", nymResponse));
```

Passo 2: Gerando uma Credencial

Nesta segunda parte da demonstração, veremos como um emissor de credenciais cria e define um *schema*, que é um documento *json* com um conjunto de atributos específicos que irão fazer parte de uma credencial. O Steward fará, portanto, uma requisição para adicionar o *schema* na *ledger*.

Criaremos então, uma credencial que faz referência ao *schema* criado e define quem irá emitir credenciais com ele e que tipo de método de assinatura é utilizado entre outras informações. Após esse processo de definição, o *Trust Anchor*, que será o emissor de credenciais no caso, usa a credencial anônima para armazená-la (*Claim Credencial*), utilizando o método de assinatura CL (Camenisch Lysychansk) de Zero *Knowledge Prove*.

```
Console.WriteLine("#9 O Emissor cria um Schema de
credencial com informações que serão requisitadas");
       String name = "gvt";
       String version = "1.1";
       String attributes =
"[\"age\",\"sex\",\"height\",\"name\"]";
       IssuerCreateSchemaResult issuerCreateSchema =
AnonCreds.IssuerCreateSchemaAsync(DidSteward.Did, name,
version, attributes).Result;
       Console.WriteLine("Schema : {0}",
issuerCreateSchema);

       Console.WriteLine("#10 Criar uma requisição para
adicionar um novo Schema na ledger");
       var schemaRequest =
Ledger.BuildSchemaRequestAsync(DidSteward.Did,
issuerCreateSchema.SchemaJson).Result;
       Console.WriteLine("Schema Request {0}",
issuerCreateSchema.SchemaJson);

       Console.WriteLine("#11 Enviar a requisição do
Schema para a ledger e obtém o resultado");
```

```
       var schemaResponse =
Ledger.SignAndSubmitRequestAsync(poolHandle,
issueWalletHandle, DidSteward.Did, schemaRequest).Result;
       Console.WriteLine("Schema Response: {0}",
schemaResponse);

       Console.WriteLine("#12 Criando e armazenando a
definição da credencial para o Schema entregue em função
do Trust Anchor");
       string configJson =
"{\"support_revocation\":false}";
       string tag = "TAG1";
       var credDef =
AnonCreds.IssuerCreateAndStoreCredentialDefAsync(issueWal
letHandle, DidTrustAnchor.Did,
issuerCreateSchema.SchemaJson, tag, null,
configJson).Result;

       Console.WriteLine("ID da Credencial:\n" +
credDef.CredDefId);
       Console.WriteLine("Definição da Credencial
JSON:\n" + credDef.CredDefJson);
```


Passo 3: Verificando uma Credencial

Nessa sessão será criado uma negociação de prova de confiança onde será verificado junto ao nó de confiança se a informação requisitada por um verificador é verdadeira com base no *schema* e na credencial gerada. Para isso, vamos gerar uma carteira para operar transações de provas de informações. Para provar que a credencial realmente pertence ao detentor, será inserido um pedaço de informação oculto chamado *link secret* na credencial.

```
Console.WriteLine("#13 Criando uma carteira para operar as
transações de prova de informação");
       string proverDid = "VsKV7grR1BUE29mG2Fm2kX";
       string proverWalletConfig =
"{\"id\":\"prover_wallet\"}";
       string proverWalletCredential =
"{\"key\":\"prover_wallet_key\"}";

       Wallet.CreateWalletAsync(proverWalletConfig,
proverWalletCredential).Wait();
       Wallet proverWalletHandle =
Wallet.OpenWalletAsync(proverWalletConfig,
proverWalletCredential).Result;

       Console.WriteLine("#14. Criação do Link Secret");
       string proverLinkSecretName = "link Secret";
       var linkSecretId =
AnonCreds.ProverCreateMasterSecretAsync(proverWalletHandle
, proverLinkSecretName).Result;

       Console.WriteLine("#15 Emissor (Trust Anchor) cria
uma oferta para o receptor da credencial");
       var credOffer =
AnonCreds.IssuerCreateCredentialOfferAsync(issueWalletHand
le, credDef.CredDefId).Result;
       Console.WriteLine("Credencial Offer : {0}",
credOffer);

       Console.WriteLine("#17 O link secret é enviado
junto a credencial requisitada");
       var credRequest =
AnonCreds.ProverCreateCredentialReqAsync(proverWalletHandl
e, proverDid, credOffer, credDef.CredDefJson,
proverLinkSecretName).Result;
       Console.WriteLine("Credencial Request: {0}",
credRequest);

       Console.WriteLine("#18 Emissor (Trust Anchor) cria
a credencial requisitada baseada no Schema");
       string credValuesJson = "{\"sex\": {\"raw\":
\"male\", \"encoded\":
```

```
\"594465709955896723921094925839488742869205008160769251991705001114423311510
3\" },"
       + "\"name\": { \"raw\": \"Alex\", \"encoded\":
\"992628570980577103383069676095884100256486223083942506668496655324486122028
74\" },"
       + "\"height\": { \"raw\": \"175\", \"encoded\":
\"175\" },"
       + "\"age\": { \"raw\": \"28\", \"encoded\": \"28\"
}}";

       var credJson =
AnonCreds.IssuerCreateCredentialAsync(issueWalletHandle,
credOffer, credRequest.CredentialRequestJson,
credValuesJson, null, null).Result;
       Console.WriteLine("Credencial Json : {0}",
credJson);

       Console.WriteLine("#19 A credencial de prova
recebida é processada e armazenada");
       var proverCredencial =
AnonCreds.ProverStoreCredentialAsync(proverWalletHandle,
null, credRequest.CredentialRequestMetadataJson,
credJson.CredentialJson, credDef.CredDefJson,
null).Result;
```

A partir de então, o emissor e a prova que recebe a credencial estabelecem uma relacionamento interativo. O emissor oferece uma credencial a um agente, que faz a requisição desta credencial enviando de forma oculta o *link secret*. Então o emissor fornece a credencial que agora estará na posse do agente em questão.

Passo 4: Autenticando Credenciais

Neste ponto, o titular gera e apresenta uma prova. Isso é feito através da construção de um JSON que seleciona as credenciais que para satisfazer a solicitação de prova com os devidos atributos. A prova é criada com a função proverCreateProof com os parâmetros apropriados. Após a validação dos atributos finalizaremos nosso exemplo efetuando fechamento e deleção das carteiras e o pool criados.

```
Console.WriteLine("#20 Obtendo a requisição de prova\n");
       string proofRequestJson = "{"
           + "\"nonce\": \"123432421212\","
           + "\"name\": \"proof_req_1\","
           + "\"version\": \"0.1\","
           + "\"requested_attributes\": {"
```



```
            + "\"attr1_referent\": {"
            + "\"name\": \"name\","
            + "\"restrictions\": [{"
            + "\"cred_def_id\": \"" + credDef.CredDefId +
"\"}]}},"
            + "\"requested_predicates\": {"
            + "\"predicate1_referent\": {"
            + "\"name\": \"age\","
            + "\"p_type\": \">=\","
            + "\"p_value\": 18,"
            + "\"restrictions\": [{"
            + "\"issuer_did\": \"" + DidTrustAnchor.Did +
"\""
            + "}]}}}";

        Console.WriteLine("Request da prova: {0}",
proofRequestJson);

        Console.WriteLine("# 21 Fazendo uma busca na
credencial pelo atributo a ser verificado(como idade por
exemplo)");
        var proverCredSearch =
AnonCreds.ProverSearchCredentialsForProofRequestAsync(pro
verWalletHandle, proofRequestJson).Result;

        string credForAttr1String =
proverCredSearch.NextAsync("attr1_referent", 1).Result;
        string referent =
JArray.Parse(credForAttr1String).First["cred_info"].Value
<string>("referent");
        Console.WriteLine(string.Format("Credencial de
prova do attr1:{0}", credForAttr1String));

        Console.WriteLine("Montando uma requisição de
prova");
        string requestedCredentialsJson = "{"
           + "\"self_attested_attributes\": {},"
           + "\"requested_attributes\": {"
           + "\"attr1_referent\": {"
           + "\"cred_id\": \"" + referent + "\","
           + "\"revealed\": true"
           + " }},"
           + "\"requested_predicates\":{"
           + "\"predicate1_referent\":{"
           + "\"cred_id\": \"" + referent + "\""
           + "}}}";

        Console.WriteLine(string.Format("Credencial
requerida para prova:{0}", requestedCredentialsJson));
```

```
       string schemasJson = new JObject(new
JProperty(issuerCreateSchema.SchemaId,
JObject.Parse(issuerCreateSchema.SchemaJson))).ToString()
;
       string credentialDefsJson = new JObject(new
JProperty(credDef.CredDefId,
JObject.Parse(credDef.CredDefJson))).ToString();

       var proofJson =
AnonCreds.ProverCreateProofAsync(proverWalletHandle,
proofRequestJson, requestedCredentialsJson, linkSecretId,
schemasJson, credentialDefsJson, "{}").Result;
       string revRegDefsJson = "{}";
       string revRegsJson = "{}";

       Console.WriteLine("Verificando se as informações
são verdadeiras");
       var valid =
AnonCreds.VerifierVerifyProofAsync(proofRequestJson,
proofJson, schemasJson, credentialDefsJson,
revRegDefsJson, revRegsJson).Result;
       Console.WriteLine("Prova :");
       Console.WriteLine("Nome= {0}",
JObject.Parse(proofJson)["requested_proof"]["revealed_att
rs"]["attr1_referent"].Value<string>("raw"));
       Console.WriteLine("Verificado= {0}", valid);
 Console.WriteLine("Fechando as carteiras");
       issueWalletHandle.CloseAsync().Wait();
       proverWalletHandle.CloseAsync().Wait();

       Console.WriteLine("Deletando as carteiras
criadas");
       Wallet.DeleteWalletAsync(issuerWalletConfig,
issuerWalletCredencial).Wait();
       Wallet.DeleteWalletAsync(proverWalletConfig,
proverWalletCredential).Wait();

       Console.WriteLine("Deletando o pool ledger");
       poolHandle.CloseAsync().Wait();
       Pool.DeletePoolLedgerConfigAsync(poolName).Wait();

   }
```


Referencias

[1] Histórias e Insumos. 'Revolução do Período Neolítico. Disponível: http://www.historiaresumos.com/revolucao-periodo-neolitica. Acessado Junho de 2019.

[2] Brealey, R., Myers, S., Allen, F. 'Princípios de Finanças Corporativas'. McGraw-Hill. 2013.

[3] IBM. '*Blockchain sharing economy*'. Disponível: https://www.ibm.com/developerworks/br/library/iot-Blockchain-sharing-economy/index.html. Acessado Junho de 2019.

[4] Tapscott, D.; Tapscott, A. '*Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*'. Maio de 2016.

[5] e-Estonia. 'e-identity solution'. Diponível: https://e-estonia.com/solutions/e-identity/id-card/. Acessado Maio 2019.

[6] Aadhaar. '*Unique Identification Authority of India*'. Disponível: https://uidai.gov.in. Acessado Julho 2019.

[7] NAKAMURA, E.T., RIBEIRO, S.L. '*Context-Based Blockchain Platform Definition and Analysys Methodology*'. The 18th International Conference on Security and Management (SAM19), Las Vegas, United States, July 2019.

[8] Cai, W., Wang, Z., Ernst, J., B., Hong, Z., Feng, C., Leung, V. C. M. '*Decentralized Applications: The Blockchain Empowered Software System*'. Disponível: https://www.researchgate.net/publication/327711685_Decentralized_Applications_The_Blockchain-Empowered_Software_System. Acessado Junho 2019.

[9] BlockchainHub. '*Blockchains & Distributed Ledger Technologies*'. Disponível: https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/. Acessado Junho 2019.

[10] Ethereum. '*How to Build a Democracy on the Blockchain*'. Disponível: https://www.ethereum.org/dao. Acessado Junho 2019.

[11] Coindesk. '*Understand the DAO Attack*'. Disponível: http://www.coindesk.com/understanding-dao-hack-journalists/. Acessado Junho 2019.

[12] Bitcoin. 'Bitcoin Core'. Disponível: https://bitcoin.org/en/bitcoin-core/. Acessado Junho 2019.

[13] RIBEIRO, S. L., NAKAMURA, E. T. 'Context-Based Blockchain Platform Definition and Analysys Methodology – Results from the application in the BlockIoT Project'. International Conference on Advances in Cyber Security, Penang, Malaysia, 2019.

[14] TechTarget. '*Details emerging on Dyn DNS DDoS attack, Mirai IoT botnet*'. Disponível: http://searchsecurity.techtarget.com/news/450401962/Details-emerging-on-Dyn-DNS-DDoS-attack-Mirai-IoT-botnet. Acessado Maio 2019.

[15] The Security Ledger. '*Mirai, The Internet of Things Bot, Goes Open Source*'. Disponível: https://securityledger.com/2016/10/mirai-the-internet-of-things-bot-goes-open-source. Acessado Maio 2019.

[16] Imperva. 'Breaking Down Mirai: An IoT DDoS Botnet Analysis'. Disponível: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html. Acessado Maio 2019.

[17] Dorri A.; Kanhere S.; Jurdak R.; Gauravaram P. '*Blockchain for IoT security and privacy: The case study of a smart home*'. IEEE. Disponível: http://ieeexplore.ieee.org/abstract/document/7917634. Acessado Maio 2019.

[18] Jun Zhou J.; Cao Z., Dong X.; Vasilakos A. '*Security and Privacy for Cloud-Based IoT: Challenges*'. IEEE. Disponível: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=7823317. Acessado Maio 2019.

[19] Fremantle P.; Aziz B.; Kirkham T. '*Enhancing IoT Security and Privacy with Distributed Ledgers - a Position Paper*'. Maio 2019.

[20] BNDES. 'Internet das Coisas: Um plano de ação para o Brasil'. Disponível: https://www.bndes.gov.br/wps/portal/site/home/conhecimento/pesquisaedados/estudos/estudo-internet-das-coisas-iot/estudo-internet-das-coisas-um-plano-de-acao-para-o-brasil. Acessado Julho 2019.

[21] McKinsey. '*Blockchain beyond the hype: What is the strategic business value?*'. Disponível: https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/blockchain-beyond-the-hype-what-is-the-strategic-business-value. Acessado Junho 2019.

[22] Jianjun, S., Jiaqi, Y., Kem Z. K. '*Blockchain-based sharing services: What blockchain technology can contribute to smart cities*'. Financial Innovation, 2:26, DOI 10.1186/ s40854-016-0040-y. 2016.

[23] McKinsey. '*Using blockchain to improve data management in the public sector*'. Disponível: https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/using-blockchain-to-improve-data-management-in-the-public-sector. Acessado Maio 2019.

[24] OWI. '*Blockchain and Identity in 2018: A Year of Promise and Pilots*'. Disponível: https://oneworldidentity.com/research/blockchain-identity-2018-year-of-promise-pilots/. Acessado Julho de 2019.

[25] Peter Steiner. '*On the Internet nobody knows you are a dog*'. Disponível: https://en.wikipedia.org/wiki/On_the_Internet,_nobody_knows_you%27re_a_dog. Acessado Julho de 2019.

[26] Cameron, K. '*The Laws of Identity*'. Microsoft Corporation. Nov 2005. Disponível: http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf. Acessado Julho 2019.

[27] Tobin, A., Reed, D. '*The Inevitable Rise of Self-Sovereign Identity*'. The Sovrin Foundation. March 2017. Disponível: https://sovrin.org/wp-content/uploads/2018/03/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf. Acessado Julho 2019.

[28] Lundkvist, C., Heck, R., Torstensson J., Mitton, Z., Sena, M. '*uPort: A Platform for Self-Sovereign Identity*'. Feb 2017. Disponível: http://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf. Acessado: Julho 2019.

[29] Ali, M., Nelson, J., Shea, R., Freedman, M. J. '*Blockstack: A Global Naming and Storage System Secured by Blockchains*'. 2016 USENIX *Annual Technical Conference* (USENIX ATC 16), Denver, CO, 2016, pp. 181–194. Disponível: https://www.usenix.org/node/196209. Acessado Julho 2019.

[30] The White House. '*National Strategy for Trusted Identities in Cyberspace: Enhancing Online Choice, Efficiency, Security, and Privacy*', Apr 2011. Disponível: https://www.hsdl.org/?view&did=7010. Acessado Julho 2019.

[31] United Nations. '*Transforming our world: the 2030 agenda for sustainable development*'. Sep 2015. Disponível: https://www.unfpa.org/resources/transforming-our-world-2030-agenda-sustainable-development. Acessado Julho 2019.

[32] Nakamoto, S. '*A Peer-to-Peer Electronic Cash System*'. Disponível: www.bitcoin.org/bitcoin.pdf. Acessado Maio 2019.

[33] ISO/IEC. '*ISO/IEC 24760-1:2019 – Information technology – Security techniques – A framework for identity management – Part1: Terminology and concepts*'. May 2019. Disponível: https://www.iso.org/standard/77582.html. Acessado Julho 2019.

[34] Zooko, W. '*Names: Distributed, Secure, Human-Readable: Choose Two*'. May 2017. Disponível: http://www.cs.princeton.edu/courses/archive/spr17/cos518/papers/zooko-triangle.pdf. Acessado Junho 2019.

[35] Hyperledger. '*Hyperledger Indy*'. Disponível: https://www.hyperledger.org/projects/hyperledger-indy. Acessado Maio 2019.

[36] W3C – DID. '*Decentralized Identifiers (DIDs)*'. Disponível: https://w3c-ccg.github.io/did-spec/. Acessado Maio 2019.

[37] Aublin, P. L., Mokhtar, S., B., Quéma, V. '*RBFT: Redundant Byzantine Fault Tolerance*'. Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on, 2013, pp. 297–306. Disponível: https://pakupaku.me/plaublin/rbft/5000a297.pdf. Acessado Maio 2019.

[38] Dunphy, P., Petitcolas, F., A., P. '*A First Look at Identity Management Schemes on the Blockchain*'. IEEE Security and Privacy Magazine. 2018.

[39] Sovrin. '*Technical Architecture Diagrams*'. Disponível: https://forum.sovrin.org/t/technical-architecture-diagrams/62/3. Acessado Junho 2019.

[40] Ethereum. '*A Next-Generation Smart Contract and Decentralized Application Platform*'. Disponível: https://github.com/ethereum/wiki/wiki/White-Paper. Acessado Maio 2019.

[41] ShoCard SITA. '*Travel Identity of the Future – White Paper*'. 2016. Disponível: https://shocard.com/wp-content/uploads/2016/11/travel-identity-of-the-future.pdf. Acessado Junho 2019.

[42] BENNET, Colin. Regulating privacy: data protection and public policy in Europe and United States. Ithaca, New York: Cornell University Press, 1992

[43] COTS, Márcio, Oliveira, Ricardo. Lei Geral de Proteção de Dados Pessoais Comentada. 1ª. Edição. São Paulo: Thomson Reuters Brasil, 2018.

[44] W3C. Decentralized Identifiers (DIDs) v0.13. Disponível em: <https://w3c-ccg.github.io/did-spec/>. Acesso em: 09/08/2019.

[45] Sovrin e Evernym. What Goes on the Ledger? Disponível em: <https://sovrin.org/wp-content/uploads/2018/10/What-Goes-On-The-Ledger.pdf>. Acesso em: 02/02/2019.

[46] LEORATTI, Alexandre. Para ministro do STJ, LGPD gera 'mais dúvidas do que certezas'. Jota, 11/12/2018. Disponível em: <https://www.jota.info/justica/lgpd-revisao-jurisprudencia-stj-11122018>. Acesso em: 02/02/2019.

[47] ZYSKING, Guy et al, 'Decentralizing Privacy: Using Blockchain to Protect Personal Data' (2015) IEEE Security and Privacy Workshops.

[48] REED, Drummond. The Story of SSI Open Standards Background on the Foundation of Self Sovereign Identity: DIDs, DKMS, DID Auth and Verifiable Credentials. 26 April 2018 SSIMeetup.org

Disponível em: <https://ssimeetup.org/story-open-ssi-standards-drummond-reed-evernym-webinar-1/>. Acesso em:092/08/2019.

[49] W3C Community Group. Decentralized Identifiers (DIDs) v0.13 - Data Model and Syntaxes. Disponível em: <https://w3c-ccg.github.io/did-spec/#introduction>. Acesso em: 09/08/2019.

[50] Hyperledger Indy. DKMS (Decentralized Key Management System) Design and Architecture V3. Disponível em: <https://github.com/hyperledger/indy-sdk/blob/677a0439487a1b7ce64c2e62671ed3e0079cc11f/doc/design/005-dkms/DKMS%20Design%20and%20Architecture%20V3.md>. Acesso em: 09/08/2019.

[51] ABNT/CEE-307. Blockchain e tecnologias de registro distribuídas – Conceitos e elementos da tecnologia Blockchain – Parte 6: Segurança, privacidade e identidade. Disponível em: <https://isolutions.iso.org/ecom/livelink/fetch/-54235805/54235807/54250561/70060171/P_307.000.000-001_Parte_06_Ago19.pdf?nodeid=70043683&vernum=-2>. Acesso em 09/08/2019.

[52] Rolf Oppliger. Microsoft .NET Passport and identity management. Information Security Technical Report, 2004.

[53] Kormann D, Rubin A. Risks of the passport single signon protocol. IEEE Computer Networks 2000.

Disponível em <https://www.cs.jhu.edu/~rubin/courses/sp03/papers/passport.pdf>. Acesso em 10/09/2019.

[54] O'Donnell, The Current and Future State of Digital Wallets. 1ª. Edição. Canadá: Creative Commons, 2019.

[55] Dan Boneh; Ben Lynn & Hovav Shacham (2004). "Short Signatures from the Weil Pairing". Journal of Cryptology.

[56] Dan Boneh, Matthew K. Franklin, Identity-Based Encryption from the Weil Pairing, SIAM J. of Computing, Vol. 32, 2003

[57] N. Theriault. Index calculus attack for hyperelliptic curves of small genus, 2003.

Disponível em <https://www.iacr.org/archive/asiacrypt2003/02_Session02/19_056/28940307.pdf>. Acesso em 10/08/2019.

[58] J-S Coron. On the Exact Security of Full Domain Hash. CRYPTO 2000. Disponível em <https://www.iacr.org/archive/crypto2000/18800229/18800229.pdf>. Acesso em 10/09/2019.

[59] D. Boneh, C. Gentry, H. Shacham, B. Lynn. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In proceedings of Eurocrypt 2003. Disponível em <https://crypto.stanford.edu/~dabo/pubs/papers/aggreg.pdf>. Acesso em 10/09/2019.

[60] Camenisch J., Lysyanskaya A. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: Yung M. (eds) Advances in Cryptology — CRYPTO 2002. Disponível em <https://link.springer.com/content/pdf/10.1007/3-540-45708-9_5.pdf>. Acesso em 10/09/2019.

Capítulo 2

Aprendizado de Máquina para Segurança: Algoritmos e Aplicações

Fabrício Ceschin[1], Luiz S. Oliveira[1], André Grégio[1]

[1]Universidade Federal do Paraná

Abstract

The massive amount of data produced by security solutions have been creating a strong dependency on automated methods for knowledge discovery. Attacks against computer systems make use of several transmission channels and formats (e.g., network traffic, binary files, text, chained system calls etc.), which difficult their observation among unsuspicious data. Machine learning techniques are a great aid for separating data into classes, but they need to be correctly deployed. In this course, we will show how to adequately apply machine learning algorithms to the security data science process. To do so, we will discuss key concepts about the subject and present practical examples with free, open source tools.

Resumo

A grande quantidade de dados produzidos por soluções de segurança criaram uma grande dependência de métodos automatizados para descoberta de conhecimento. Ataques contra sistemas computacionais utilizam diversos canais de transmissões e formatos (por exemplo, tráfego de rede, arquivos binários, chamadas de sistema encadeadas, etc) que dificultam a sua observação em meio à dados não suspeitos. Técnicas de aprendizado de máquina são uma grande ajuda para separar dados em classes, mas elas precisam ser implantadas corretamente. Neste curso, nós mostraremos como aplicar corretamente algoritmos de aprendizado de máquina ao processo de ciência dos dados em segurança. Para isso, discutiremos conceitos chave sobre o assunto e apresentaremos exemplos práticos com ferramentas grátis e open source.

2.1. Introdução

Segundo relatório da Symantec [Symantec 2019], cerca de 144 milhões de ataques foram bloqueados na plataforma Windows somente em 2018, uma média de 12 milhões por mês. Essa grande quantidade de ataques cibernéticos em um curto espaço de tempo trás a necessidade de se utilizar métodos inteligentes para análise dos mesmos, tais como sistemas baseados em inteligência artificial, capazes de aprender com os dados recebidos (aprendizado de máquina). O aprendizado de máquina é a ciência de programar computadores para que eles aprendam através de dados para resolver uma determinada tarefa, cuja performance pode ser avaliada através de uma métrica [Gron 2017]. Os dados coletados geralmente pertencem a grupos pré-estabelecidos, cujo agrupamento se dá pela semelhança de seus padrões. No caso de ataques cibernéticos, por exemplo, consideramos a detecção de *malware* como sendo o problema (ou tarefa) composto por duas classes: *malware* (programas malignos) e *goodware* (programas benignos). A semelhança entre os *malware* se dá pelo fato de que esse tipo de programa realiza alguma ação maliciosa cujo objetivo é causar algum dano, alterar ou até mesmo roubar dados. Já os *goodware* são programas cujo comportamento é esperado, sem causar nenhum dos problemas já mencionados. A partir desses dados coletados e das classes definidas, um algoritmo é treinado para identificar se um determinado *software* desconhecido é ou não um *malware*. Para avaliar esse algoritmo treinado (modelo), existem diversas métricas que medem o quão bom é o sistema como um todo, sendo que cada métrica pode apresentar uma perspectiva diferente do problema.

A área de segurança computacional está repleta de resultados de pesquisa e desenvolvimento dependentes de algoritmos de aprendizado de máquina, seja na academia ou na indústria. Entretanto, ao se avaliar os artigos científicos, *white-papers* e até mesmo apresentações de empresas que fazem uso de aprendizado de máquina na detecção ou classificação de ataques, observa-se que não raro as aplicações de tais algoritmos contém problemas, o que acarreta em resultados muitas vezes irreais. A extensão dos problemas varia da coleta e pré-processamento dos dados ao treinamento, classificação e interpretação das informações obtidas. Em problemas de classificação de imagem, por exemplo, os dados possuem distribuição estacionária, isto é, um cachorro sempre será um cachorro e um gato sempre será um gato, isso nunca muda e os modelos assumem que é sempre assim. Entretanto, em segurança, os dados nem sempre possuem essa distribuição: os dados geralmente tem distribuição não-estacionária e mudam conforme o tempo passa, seja para se adaptar a mudanças ou para se aproveitar de falhas desconhecidas (*zero-day*), nos casos de *malware* [Ceschin et al. 2018].

Neste curso, tem-se por objetivo principal promover a aplicação adequada de algoritmos de aprendizado de máquina em dados de segurança, respeitando as etapas básicas necessárias na ciência de dados. Para tanto, visa-se familiarizar o participante com os algoritmos clássicos de aprendizado de máquina e seus usos tradicionais na classificação/detecção e agrupamento de dados que representem instâncias normais/benignas e anômalas/maliciosas. No decorrer do curso, cada etapa do processo de ciência de dados será abordada, desde a coleta, seleção de atributos, escolha do algoritmo, treinamento, avaliação, validação e interpretação dos resultados. Em cada etapa, serão mostradas ferramentas *open source* que auxiliem no cumprimento da tarefa em questão. Todos os códigos fonte e conjunto de dados utilizados neste curso estão disponíveis em um repositório no

github[1].

Este capítulo está organizado da seguinte forma: a Seção 2.2 apresenta como realizar a coleta de dados e rotulá-los de forma correta, destacando os principais problemas relacionados e exemplos de *datasets* (que são utilizados ao longo do curso); na Seção 2.3 destacamos métodos de extração de atributos estáticos e dinâmicos de arquivos desenvolvidos para Windows e Android; a extração de características (utilizando TF-IDF e *Word2Vec*) e normalização das mesmas é o foco da Seção 2.4; a Seção 2.5 apresenta os modelos mais utilizados na literatura de segurança e aprendizado de máquina, dentre eles classificadores, detectores de mudança e agrupadores, incluindo exemplos práticos de como utilizá-los corretamente, bem como as principais bibliotecas; a avaliação dos modelos é o destaque da Seção 2.6, com as principais métricas e modos de validação; na Seção 2.7 apresentamos nossas conclusões.

2.2. Datasets

O conjunto de dados utilizados por um algoritmo de aprendizado de máquina, chamado de *dataset*, é fundamental para o processo de aprendizagem, uma vez que todo algoritmo irá se basear nos exemplares apresentados (conjunto de treino) para tomar uma decisão conforme a tarefa utilizada. Neste minicurso, definimos que um *dataset* pode ser disponibilizado de três formas, cada uma delas relacionada a uma etapa que antecede a criação de um modelo:

1. **Dados Brutos:** os dados brutos são disponibilizados como foram coletados. Um exemplo seriam os executáveis de *malware* em diferentes formatos, tais como PE, ELF e APK, ou tráfego de rede capturado em formato PCAP.

2. **Atributos:** metadados extraídos (geralmente após a coleta e pré-processamento) dos dados brutos são disponibilizados de maneira filtrada, isto é, com menos ruído e mais foco no dado que realmente importa. Um exemplo são *logs* de execução de um software específico ou dados extraídos de seu cabeçalho, ou informações resumidas acerca de um subconjunto do tráfego de rede coletado.

3. **Características:** dados inatos que distinguam amostras, já extraídas dos atributos coletados e processados e prontas para serem utilizadas como entrada no classificador. Um exemplo seria a transformação de *logs* em um vetor para cada software mencionado no passo anterior, cujas posições desse vetor correspondem às características. Outro exemplo seria um arquivo de tráfego que representa um conjunto de origens e/ou destinos de interesse, no qual tenta-se caracterizar cada interação por meio da discretização ou frequência de atributos pré-definidos (ex., protocolos, quantidade de pacotes, bytes, etc.).

Nesta seção focaremos mais em como coletar corretamente os dados para construir um *dataset* e exemplificaremos como rotular um conjunto de dados e quais os problemas comuns que podem acontecer, como a falta de classes para rotulação ou o desbalanceamento das classes. Serão também apresentados *datasets* disponíveis na literatura que

[1] `https://github.com/fabriciojoc/ml-cybersecuritiy-course`

podem ser utilizados para melhor entendimento dos conceitos mostrados nesse cursos, tais como conjuntos de programas maliciosos, mensagens de e-mail não-solicitadas, tráfego de rede e registros de auditoria diversos.

2.2.1. Coletando Dados

A coleta de dados pode ser uma das etapas mais desafiadoras de um projeto, uma vez que todo o problema se baseia nesses dados e pode variar muito de acordo com o domínio e origem dos mesmos. Muitas vezes os dados podem ser provenientes de fontes sensíveis, que não permitem que os dados sejam compartilhados ou utilizados de determinada forma, o que pode acabar comprometendo a reprodutibilidade de uma solução. Se os dados envolvem informações pessoais ou restritas, manter a privacidade dessas informações é necessária (recomenda-se utilizar técnicas de privacidade diferencial nesses casos [Dwork and Roth 2014]). Além disso, recomenda-se coletar o maior número de informações úteis (aquelas que tendem a diferenciar um exemplo de uma classe de outra) possíveis, de forma que ocupe o menor espaço em disco (o que geralmente pode ser uma limitação). Uma informação fundamental que deve-se coletar é o tempo (data de primeira aparição, coleta, desenvolvimento, etc): dados relacionados a segurança geralmente fazem parte de uma distribuição não-estacionária, isto é, não possuem sempre a mesma distribuição e estão em constante mudança, seja para evadir as soluções de segurança ou para se adequar às atualizações de hardware e software [Ceschin et al. 2018]. Nas subseções a seguir trataremos de alguns problemas relacionados a coleta de dados.

2.2.2. Rotulando Dados

Dependendo do domínio do problema, rotular os dados pode ser difícil. Particularmente, os problemas envolvendo *malware* são desafiadores, já que os rótulos são baseados em anti-vírus que usam detecção por assinatura e muitas vezes não possuem o rótulo correto logo quando uma ameaça é encontrada (ela é apenas marcada como uma ameaça para que seja detectada, mas geralmente não é classificada corretamente em sua família desde o início). Tal fato pode complicar a remediação após um ataque, uma vez que dependendo da família do *malware*, diferentes ações podem ser tomadas para voltar o sistema ao estado normal. Por isso, ao rotular os dados de programas maliciosos, recomendamos utilizar a API do VirusTotal[2] em conjunto com a ferramenta AVClass [Sebastián et al. 2016]. Através do VirusTotal, pode-se obter um relatório completo de um determinado arquivo, com o resultado de cada análise dos anti-vírus disponíveis na plataforma. O Código 1 apresenta um exemplo de como obter um relatório utilizando a API (é necessário fornecer sua própria chave de API ao enviar a requisição). Após criar um arquivo JSON (no nosso caso, "samples.json") em que cada linha é um relatório de cada exemplar do **dataset**, deve-ser clonar o repositório do AVClass[3] e executar o comando mostrado no Código 2 (em um terminal), que resultará nos seguintes arquivos:

- **samples.verbose:** criado devido ao parâmetro -v", contém todas as famílias extraídas para cada exemplo, ranqueadas com o número de anti-vírus que o classificaram como determinada família;

[2] `https://developers.virustotal.com/reference`
[3] `https://github.com/malicialab/avclass`

- **samples.families:** criado devido ao parâmetro -fam", contém uma tabela que conta o número de exemplares por família do arquivo de entrada;

- **samples.labels:** criado como saída, contém a família correspondente de cada exemplo de entrada.

```
import requests
# url de relatorio do virus total
url = 'https://www.virustotal.com/vtapi/v2/file/report'
# chave de api (altere para a sua chave)
API_KEY = "<api_key>"
# parametros: apikey e a chave de api; resource e o
  ↪ md5/sha1/sha256 do arquivo
params = {'apikey': API_KEY, 'resource':
  ↪ '6545c6f328393f9b3168e260ae1b7135c1bfa917'}
# envia requisicao e pega a resposta
response = requests.get(url, params=params)
# salva resposta em um arquivo .json
with open('samples.json', 'w') as out:
    json.dump(response.json(), out)
```

```
{'md5': 'ab243d1fad9ed1af747f313ec88b3fd0',
 ...
 'scans': {
  ...,
  'AVG': {'detected': True,
   'result': 'FileRepMalware',
   'update': '20180325',
   'version': '18.2.3827.0'},
  ...,
  'nProtect': {'detected': False,
   'result': None,
   'update': '20180325',
   'version': '2018-03-25.01'}},
 ...
 'total': 67,
 'verbose_msg': 'Scan finished, information embedded'}
```

Código 1. Obtendo relatório de um exemplar no VirusTotal.

Com o arquivo de saída gerado é possível relacionar os dados originais e obter a família de *malware* correspondente para cada exemplar do *dataset* criado, ”normalizando“ os resultados gerados pelo VirusTotal.


```
./avclass_labeler.py -vt samples.json -v -fam >
  ↪ samples.labels
```

```
ab243d1fad9ed1af747f313ec88b3fd0        scriptkd
```

Código 2. Normalizando um arquivo JSON com relatórios do VirusTotal, criando um único arquivo com a respectiva família de cada exemplar.

2.2.3. Problemas

Ao coletar dados e criar um *dataset*, estamos propensos a diversos problemas que podem afetar diretamente a qualidade dos resultados produzidos. Dentre eles, os seguintes:

- **Dados Insuficientes:** as vezes a quantia de dados coletadas pode não ser o suficiente para que o modelo de aprendizado de máquina generalize o problema de forma adequada. Para problemas complexos, tais como reconhecimento de imagens ou fala, podem ser necessárias milhões de imagens [Gron 2017].
- **Dados Não Representativos:** é importante que os dados coletados representem de fato o problema, incluindo novos casos que devem ser generalizados. Caso contrário, a classificação pode apresentar resultados muito ruins, sobretudo ao ser aplicado no mundo real [Gron 2017].
- **Dados com Pouca Qualidade:** dados com erros, *outliers* e ruídos dificultam a identificação de padrões, diminuindo o desempenho da classificação [Gron 2017].
- **Atributos Irrelevantes:** um sistema de aprendizado de máquina será capaz de aprender através de atributos e características relevantes, isto é, aquelas que conseguem diferenciar efetivamente uma classe de outra. Não há segredo: se houver lixo na entrada do sistema, qualquer algoritmo de aprendizado de máquina terá lixo como saída (conhecido como *garbage in, garbage out*) [Gron 2017]. Por isso, a seleção de atributos é tão importante quanto a extração de características e algoritmo de classificação.
- **Falta de Classes:** a falta de classes pode ser um problema dependendo da fonte dos dados coletados: as vezes eles são filtrados (e não representam o problema real) ou até mesmo parte das classes não aparecem onde foram coletadas. Por exemplo, se os dados forem coletados em um país em que determinado ataque não acontece e a solução for aplicada em outro país em que ele acontece, não haverá nenhum exemplo dessa classe para que possa ser feita a classificação.
- **Desbalanceamento de Classes:** muito comum em problemas de segurança, uma vez que determinados tipos de ataques acontecem mais frequentemente que outros, o que é um problema para o classificador, já que pode não generalizar o suficiente as classes menos presentes e ter um viés a favor das classes que aparecem mais.
- ***Delay* de Classes:** a classe de determinado exemplar pode não estar disponível na mesma hora em que o mesmo foi coletado, afetando na atualização do classificador

ao obter novos dados, já que o mesmo só poderá conferir se classificou corretamente assim que a classe dele estiver disponível.

2.2.4. Exemplos

Neste curso usaremos dois *datasets* que já foram utilizados em trabalhos anteriores na literatura [Arp et al. 2014, Ceschin et al. 2018] de classificação de *malware*, um de Windows e outro de Android. Ambos estão no formato *CSV* e já contém os atributos extraídos dos dados coletados, o primeiro (de Windows) coletado por nós mesmos e o segundo (Android), melhorado com as informações temporais de cada exemplar, informação crucial para a avaliação correta de um modelo de aprendizado de máquina para segurança.

2.2.4.1. *Brazilian Malware*

O *dataset Brazilian Malware* contém atributos de 50.181 exemplares de *goodware* e *malware* de Windows (do tipo Portable Executable – PE) coletados entre os anos de 2013 e 2019 (21.116 *goodware* e 29.065 *malware*). Os exemplares de *goodware* foram coletados utilizando um *web crawler* em três sites de *download*: *Sourceforge*[4], *Softonic*[5] e *CNET Download*[6]. Os exemplares de *malware* foram obtidos de uma instituição financeira nacional que prefere se manter anônima, advindos de mecanismos de segurança instalados em seus clientes ou coletados de *phishing*. Até o momento deste trabalho, foram coletados $\approx$ 50 GB de binários de *malware*, porém, como a disponibilização de *malware* é proibida no Brasil (e a disponibilização dos mesmos não pode ser feita devido a restrições da instituição financeira), a disponibilidade do *dataset* só pôde ser feita através de atributos (no caso, extraímos atributos estáticos) no formato *CSV*. Além da *hash* SHA1 de cada exemplar, o *dataset* contém as informações dos cabeçalhos dos arquivos *PE* (de forma similar ao apresentado na Seção 2.3.1.1), bem como a entropia do arquivo, bibliotecas, funções importadas e lista de nomes de empacotadores, compiladores ou ferramentas usadas.

2.2.4.2. Drebin

O *dataset* Drebin foi originalmente disponibilizado em 2014, apresentando uma solução para detecção de *malware* no próprio *smartphone*, com taxa de detecção de 94% e baixa taxa de falso positivo, levando cerca de dez segundos em média para analisar um aplicativo [Arp et al. 2014]. O *dataset* contém atributos estáticos extraídos (de forma muito similar ao apresentado na Seção 2.3.1.2, com permissões, funções, URLs, atividades, etc) de 129.013 aplicativos de Android coletados entre 2010 e 2012, sendo 5.560 deles *malware* e 123.453, *goodware*. Como o *dataset* original não possuía informações temporais dos exemplares, nós desenvolvemos uma melhoria do mesmo com esses dados extraídos do Virus Total, considerando que a data de um exemplar é a sua primeira aparição no sistema, o que acabou gerando uma inconsistência, já que alguns exemplares apareceram antes no Virus Total (2009) e alguns, apenas depois (2013 e 2014).

[4] `https://sourceforge.net/`
[5] `https://en.softonic.com/`
[6] `https://download.cnet.com/windows/`

2.3. Obtenção de Atributos

Atributos são fundamentais para o processo de aprendizagem, uma vez que, após terem suas características extraídas e serem rotulados adequadamente, são utilizados como entrada para o treinamento de um classificador. Neste curso, a obtenção de atributos será feita em amostras de programas de Windows e Android por meio da apresentação de ferramentas livres que implementem dois tipos de análise: estática e dinâmica [Gandotra et al. 2014].

2.3.1. Análise Estática

A análise estática consiste em extrair atributos de um *software* sem executá-lo. Geralmente, extraem-se strings de assinaturas, sequência de bytes, chamadas de sistemas, grafo de fluxo de controle, bibliotecas, etc [Gandotra et al. 2014]. Neste curso, abordaremos a extração de atributos estáticos de arquivos *Portable Executable* (PE) e *Android Package Kit* (APK) nas Subseções a seguir.

2.3.1.1. *Portable Executable* (PE)

Para extrair atributos estáticos de arquivos do tipo *Portable Executable* (PE), usaremos a biblioteca *pefile*[7]. A maioria das informações contidas nos cabeçalhos do arquivo PE, seções e seus dados podem ser facilmente acessadas através dessa biblioteca [Yonts 2010, Saxe and Sanders 2018]. O Código 3 apresenta um exemplo de como abrir um arquivo PE utilizando um arquivo executável.

```
1 import pefile
2 # localizacao do arquivo
3 file_location = "./datasets/samples/pe/WinRAR.exe"
4 # abre arquivo
5 pe = pefile.PE(file_location)
```

Código 3. Abrindo um arquivo PE utilizando a biblioteca `pefile`.

Após abrir o arquivo, é possível obter facilmente seus atributos, como mostrado no Código 4, em que as informações do cabeçalho são impressas. No total, são sete atributos que podem ser extraídos: `Machine`, `NumberOfSections`, `TimeDateStamp`, `PointerToSymbolTable`, `NumberOfSymbols`, `SizeOfOptionalHeader` e `Characteristics`. Note que é possível obter cada atributo separadamente ao acessar a variável `FILE_HEADER` do objeto `pe`. Ao mesmo tempo, também é possível obter atributos do cabeçalho opcional do arquivo, basta acessar a variável `OPTIONAL_HEADER` do objeto `pe`.

Além dos atributos do cabeçalho do arquivo, também é possível obter as bibliotecas dinâmicas importadas pelo mesmo. Para isso, basta acessar a variável `DIRECTORY_ENTRY_IMPORT` do objeto `pe`. Esta variável mapeia cada *DLL* utilizada pelo

[7] `https://github.com/erocarrera/pefile`

```
# obtem atributos do header do arquivo pe
print(pe.FILE_HEADER)
# obtem numero de sessoes do arquivo (para qualquer
  ↪ outro atributo, basta alterar "NumberOfSections")
print(pe.FILE_HEADER.NumberOfSections)
```

```
[IMAGE_FILE_HEADER]
0x114   0x0   Machine:              0x8664
0x116   0x2   NumberOfSections:     0x8
0x118   0x4   TimeDateStamp:        0x5C72EA4B [Sun Feb
  ↪ 24 19:02:35 2019 UTC]
0x11C   0x8   PointerToSymbolTable: 0x0
0x120   0xC   NumberOfSymbols:      0x0
0x124   0x10  SizeOfOptionalHeader: 0xF0
0x126   0x12  Characteristics:      0x22
8
```

Código 4. Obtendo os atributos do cabeçalho do arquivo PE.

programa a um objeto, cujo nome da biblioteca pode ser acessado através da variável `dll`. Um exemplo pode ser visto no Código 5. Através dessa mesma variável, é possível obter todas as funções chamadas por cada biblioteca utilizada pelo programa, como mostrado no Código 6. Mais exemplos de como obter mais dados do arquivo PE podem ser encontrados no repositório do curso. Também recomendamos a leitura deste artigo[8], que também apresenta exemplos interessantes de como utilizar a biblioteca `pefile`.

```
# lista de dlls:
dlls = []
# caminha em DIRECTORY_ENTRY_IMPORT
for d in pe.DIRECTORY_ENTRY_IMPORT:
    # adicionar dll atual na lista
    dlls.append(d.dll)
# imprime lista
print(dlls)
```

```
['KERNEL32.dll', 'USER32.dll', 'GDI32.dll',
  ↪ 'COMDLG32.dll', 'ADVAPI32.dll', 'SHELL32.dll',
  ↪ 'ole32.dll', 'OLEAUT32.dll', 'SHLWAPI.dll',
  ↪ 'POWRPROF.dll', 'COMCTL32.dll', 'UxTheme.dll',
  ↪ 'gdiplus.dll', 'MSIMG32.dll']
```

Código 5. Obtendo bibliotecas dinâmicas do arquivo PE.

[8] https://axcheron.github.io/pe-format-manipulation-with-pefile/


```
# lista de simbolos
symbols = []
# caminha em DIRECTORY_ENTRY_IMPORT
for i in pe.DIRECTORY_ENTRY_IMPORT:
    # caminha nas funcoes da biblioca que sao importadas
    for s in i.imports:
        # verifica se o simbolo e valido
        if s.name != None:
            # adiciona a lista de simbolos
            symbols.append(s.name)
# imprime simbolos
print(symbols)
```

```
['DeviceIoControl', 'BackupRead', 'BackupSeek',
 ↪ 'GetShortPathNameW', 'GetLongPathNameW',
 ↪ 'GetFileType', 'GetStdHandle', 'FlushFileBuffers',
 ↪ 'GetFileTime', 'GetDiskFreeSpaceExW',
 ↪ 'GetVersionExW', ... , 'GradientFill']
```

Código 6. Obtendo funções importadas pelo arquivo PE.

2.3.1.2. *Android Package Kit* (APK)

Quando se trata de arquivos do tipo *Android Package Kit* (APK), pode-se utilizar a biblioteca *androguard*[9] para obter atributos estáticos. Com ela é possível obter informações do manifesto do aplicativo, seus recursos, arquivos DEX descompilados e muito mais. Similar ao método utilizado para arquivos PE, primeiramente abre-se o arquivo APK, como mostrado no código 7 (a leitura do APK pode demorar, dependendo do seu tamanho). A diferença aqui se dá ao fato de que a biblioteca nos retorna três objetos: um da classe `APK` (que provê todas as informações do APK), outro um vetor com objetos da classe `DalvikVMFormat` (que corresponde ao arquivo DEX encontrado dentro do APK) e o último da classe `Analysis` (que contém classes especiais para lidar com aplicativos multi-DEX). Neste curso focaremos apenas no primeiro objeto afim de exemplificar a coleta de atributos. Para mais detalhes mais aprofundados sobre a biblioteca, recomenda-se a leitura da documentação completa[10].

[9] `https://github.com/androguard/androguard`
[10] `https://androguard.readthedocs.io/en/latest/index.html`

```
from androguard.misc import AnalyzeAPK
# localizacao do arquivo
file_location =
  ↪ "./datasets/samples/apk/com.whatsapp_2.19.203-
  ↪ 452877_minAPI15(armeabi-
  ↪ v7a)(nodpi)_apkmirror.com.apk"
# abre arquivo
a, d, dx = AnalyzeAPK(file_location)
```

Código 7. Abrindo um arquivo APK usando a biblioteca `androguard`.

Após a leitura do arquivo, utilizando os objetos retornados, pode-se obter as permissões e as atividades que estão atrelados ao *AndroidManifest.xml*, como demonstrado nos Códigos 8 e 9. Com o mesmo objeto, é possível também obter as versões do aplicativo (tanto o valor númerico como a string da versão) e do SDK mínimo, máximo e versão alvo, como apresentado no Código 10.

```
print(a.get_permissions())
```

```
['android.permission.AUTHENTICATE_ACCOUNTS',
  ↪ 'android.permission.CHANGE_WIFI_STATE',
  ↪ 'android.permission.ACCESS_FINE_LOCATION',
  ↪ 'android.permission.SEND_SMS', ...,
  ↪ 'android.permission.MANAGE_OWN_CALLS']
```

Código 8. Obtendo permissões utilizadas pelo APK.

```
print(a.get_activities())
```

```
['org.npci.commonlibrary.GetCredential',
  ↪ 'com.whatsapp.WebSessionsActivity',
  ↪ 'com.whatsapp.VoiceMessagingActivity',
  ↪ 'com.whatsapp.IdentityVerificationActivity', ...,
  ↪ 'com.google.android.gms.common.api.GoogleApiActivity']
```

Código 9. Obtendo atividades do APK.

2.3.2. Análise Dinâmica

A análise dinâmica consiste em extrair atributos de um *software* executando o mesmo em um ambiente controlado (máquina virtual, simulador, emulador, *sandbox*, etc). Durante a execução, é comum monitorar as chamadas de sistemas e seus parâmetros, fluxo de informação, traços de execução, recursos utilizados, etc [Gandotra et al. 2014]. No Windows, existem diversas estratégias utilizadas para analisar dinamicamente um *software* na


```
print(a.get_androidversion_code())
print(a.get_androidversion_name())
print(a.get_min_sdk_version())
print(a.get_max_sdk_version())
print(a.get_target_sdk_version())
```

```
452877
2.19.203
15
None
28
```

Código 10. Obtendo versões do aplicativo (numérico e *string*) e do SDK (mínimo, máximo e versão alvo) utilizados.

literatura [Botacin et al. 2018]. Dentre elas, uma das mais utilizadas é o *Cuckoo Sandbox*[11], uma ferramenta *open source* que automatiza todo o processo de análise dinâmica de diversos arquivos para o Windows, apresentando todo o seu traço de execução (arquivos lidos e alterados, chaves de registros alteradas, etc), tráfego de rede e até mesmo as alterações feitas em memória [Oktavianto and Muhardianto 2013]. Como não é o foco do curso, fica a cargo do leitor ler a documentação referente a obtenção desses dados caso haja interesse, que geralmente precisa de uma extensa lista de passos para ser executada. Também recomendamos a leitura de um minicurso já apresentado no SBSEG de 2011 [Filho et al. 2011]. Já no Android, o tema também é muito extenso e complexo. Por isso recomendamos a leitura dos *surveys* [Hoffmann et al. 2016, Tam et al. 2017] para mais informações sobre os métodos de análise dinâmica desta plataforma.

Relatamos aqui também que é possível obter relatórios de execução dinâmica através do VirusTotal, tanto de Android como de Windows. Entretanto, essa função faz parte da API privada (que é paga). Um exemplo de relatório de *software* do Windows é o Código 11, um *JSON* extraído do VirusTotal (que utiliza uma versão um pouco modificada do *Cuckoo Sandbox*), contendo diversas informações referentes ao comportamento do exemplar, tais como funções executadas (e seus parâmetros) e *hosts* utilizados no tráfego de rede (devido ao espaço ocupado pelo *log* original, compactamos sua representação. Mais informações podem ser obtidas no arquivo original[12]). Um exemplo de análise dinâmica de Android do VirusTotal pode ser visto nesta página[13].

[11] https://cuckoo.sh/docs/
[12] https://tinyurl.com/y36uxamz
[13] https://tinyurl.com/y25wkow6

```
{ "info":{ "started":"2013-02-27 14:44:31",
  ↪ "duration":"15 seconds", "version":"v0.1",
  ↪ "ended":"2013-02-27 14:44:46" },
    "network":{ "hosts":["0.0.0.0", "255.255.255.255",
     ↪ "10.0.2.2", "10.0.2.15", "239.255.255.250",
     ↪ "224.0.0.22", "65.55.21.14", "10.0.2.255"] },
    "behavior":{
       "processes":[
          {  "parent_id":"1940",
             "process_id":"2000",
             "process_name":"6c7a2a4dae13df742a60c0f...",
             "first_seen":"20130227134444.940",
             "calls":[
                {  "category":"device",
                 ↪ "status":"SUCCESS",
                   "return":"",
                   "timestamp":"20130227134444.940",
                   "repeated":6, "api":"DeviceIoControl",
                   "arguments":[
                      { "name":"hDevice",
                       ↪ "value":"0x00000044" },
                      { "name":"dwIoControlCode",
                       ↪ "value":"0x00390008" },
                      { "name":"lpInBuffer",
                       ↪ "value":"0x77e46318" },
                      { "name":"nInBufferSize",
                       ↪ "value":"0x00000100" },
                      { "name":"lpOutBuffer",
                       ↪ "value":"0x0012fbbc" },
                      { "name":"nOutBufferSize",
                       ↪ "value":"0x00000100" },
                      { "name":"lpBytesReturned",
                       ↪ "value":"0x0012fbb4" },
                      { "name":"lpOverlapped",
                       ↪ "value":"0x00000000" } ]
                } ] } ],
       "summary":{
          "files":[ "C:\\6c7a2a4dae13df742a60c0f..." ]
} } }
```

Código 11. Exemplo de *log* em *JSON* produzido pelo *Cuckoo Sandbox* do VirusTotal.

2.4. Extração de Características

Muitas vezes os atributos extraídos dos dados não podem ser utilizados diretamente em um modelo de aprendizado de máquina. Nos atributos apresentados na Seção anterior, por exemplo, apenas os atributos numéricos (cujo valor são números reais ou inteiros) poderiam ser utilizados diretamente em um modelo. Já os vetores de palavras extraídas, tais como bibliotecas e funções utilizadas, devem passar por uma etapa a mais chamada de extração de características, cujo objetivo é transformar esses atributos em algo que o classificador consiga "entender", geralmente definindo um mesmo número de características como saída desse processo (já que um programa pode usar mais bibliotecas que outros, por exemplo). Nesta Seção, explicaremos dois métodos de extrair características de textos (conjuntos de palavras, como as extraídas anteriormente). Além disso, também explicaremos como normalizar essas características após a extração.

2.4.1. Métodos

Apresentaremos dois métodos de extração de características de textos, ambos muito utilizados na literatura: TF-IDF e *Word2Vec*. Dado um texto representando um atributo, ambos os métodos produzem como saída um vetor numérico representando de uma forma diferente esse texto. Vale ressaltar que, no caso dos exemplos dados na Seção 2.3, os vetores de palavras extraídos são transformados em textos em que cada palavra é separada por um espaço.

2.4.1.1. TF-IDF

Dado um vocabulário de um conjunto de documentos, isto é, todas as palavras que aparecem neste conjunto, cada documento i é representado por um vetor $\vec{d_i} = (w_{i,1}, w_{i,2}, ..., w_{i,t})$, em que $w_{i,j}$ representa o *TF-IDF* (do inglês *Term Frequency—Inverse Document Frequency*) da *j-ésima* palavra do vocabulário. O *TF-IDF* é uma medida estatística utilizada para avaliar o quão importante uma palavra é para um documento em relação à uma coleção de documentos [Manning et al. 2008a]. Esta medida é obtida através da multiplicação de dois termos:

- *Term Frequency (TF)*: mede com que frequência uma palavra/termo t ocorre em um texto/documento, i.e.,

$$TF(t) = \frac{\text{Número de vezes que } t \text{ aparece no documento}}{\text{Número total de palavras do documento}} \tag{1}$$

- *Inverse Document Frequency (IDF)*: mede o quão importante é uma termo t, i.e.,

$$IDF(t) = \log_e \left(\frac{\text{Número total de documentos}}{\text{Número de documentos com a palavra } t} \right) \tag{2}$$

Resumidamente, cada texto/documento é representado por um vetor esparso que contém suas medidas de *TF-IDF* para cada palavra do vocabulário. O vocabulário pode

ser reduzido a um número V de palavras, sendo essas as que mais estão presentes nos documentos/textos.

Utilizando o *dataset Brazilian Malware* (apresentado na Seção 2.2.4.1), o Código 12 apresenta uma forma de extrair características das listas de bibliotecas utilizadas por cada exemplar com o TF-IDF, através da classe `TfidfVectorizer`[14] da biblioteca scikit-learn [Pedregosa et al. 2011]. Primeiramente, lê-se o *dataset* e inicializamos o extrator de características, no caso com um número máximo de 200 características no total (cada característica representa uma palavra, conforme já mencionado). Então, dividimos os dados de bibliotecas importadas ao meio e utilizamos a primeira metade para treinar o extrator (através do método `fit`), gerando o dicionário de palavras a ser extraído. Finalmente, transformam-se os textos de treino e teste em características (através do método `transform`). Pela saída, pode-se observar que o número de colunas da matriz resultante é 200, por conta do máximo de características que foi configurado ao criar o extrator.

```
from sklearn.feature_extraction.text import
  ↪ TfidfVectorizer
import pandas as pd
# local do dataset
data_path = "./datasets/brazilian-malware.csv"
# le dataset em CSV
data = pd.read_csv(data_path)
# obtem lista de dlls
texts = data["ImportedDlls"].values
# obtem o meio dos dados
mid = int((len(texts) + 1)/2)
# divide dados em treino e teste
train_texts = texts[:mid]
test_texts = texts[mid:]
# inicializa tfidf com maximo de 200 features
vectorizer = TfidfVectorizer(max_features=200)
# treina tfidf
vectorizer.fit(train_texts)
# transforma textos para caracteristicas
train_features = vectorizer.transform(train_texts)
test_features = vectorizer.transform(test_texts)
# imprime a forma das caracateristicas
print(train_features.shape, test_features.shape)
```

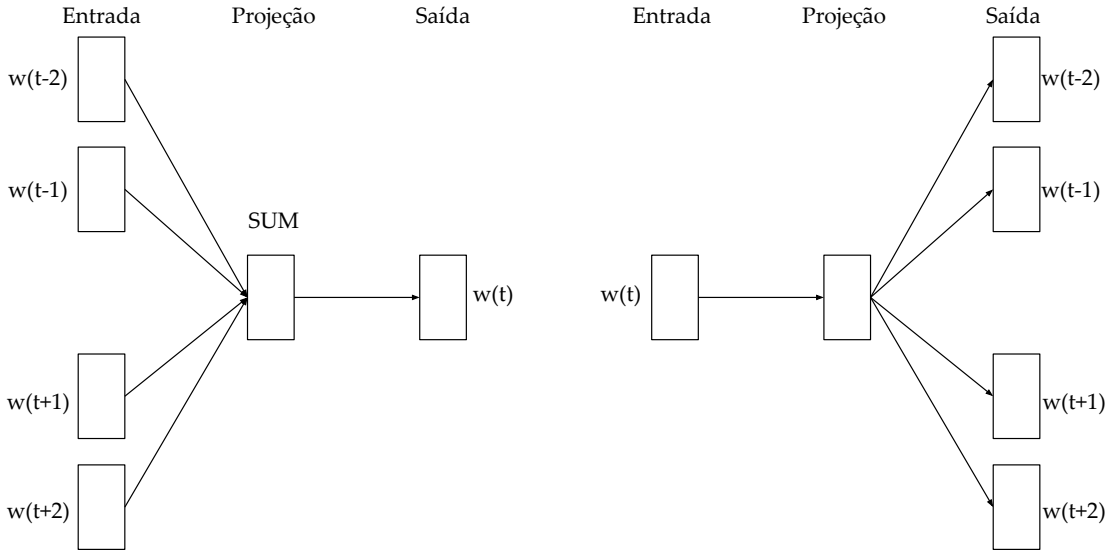
```
(25091, 200) (25090, 200)
```

Código 12. Utilizando TF-IDF para extrair características de uma lista de bibliotecas do *dataset Brazilian Malware*.

[14] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

2.4.1.2. Word2Vec

Desenvolvido em 2013 por pesquisadores do *Google* [Mikolov et al. 2013a], o *word2vec* é uma forma de criar vetores que representam palavras. A maior inovação desta proposta é a capacidade de representar a semelhança entre palavras e seus significados, diferente do *TF-IDF*, que apenas projeta os textos baseando-se na frequência das mesmas. Além disso, o *word2vec* projeta somente as palavras e não o texto, sendo necessário utilizar alguma estratégia para se extrair características dessas palavras projetadas. O modelo geral do *word2vec* utiliza uma rede neural não supervisionada do tipo *feed-forward* (para mais detalhes sobre rede neural, checar a Seção 2.5.1.4). Existem duas arquiteturas disponíveis neste modelo: *continuous bag-of-words (CBOW)* e *skip-gram*. A primeira (*CBOW*) tenta predizer a palavra central de um conjunto de palavras ao seu redor (entrada), semelhante a um *cluster*. Em contrapartida, a arquitetura *skip-gram* faz o inverso: tenta predizer as palavras ao redor, dado uma palavra central como entrada. A Figura 2.1 apresenta ambos os modelos [Shulman 2016, Mikolov et al. 2013b]. Como apenas a arquitetura *CBOW* foi utilizada neste trabalho, ela será explicada em detalhes a seguir.



(a) CBOW. (b) Skip-gram.

Figura 2.1. Arquiteturas do *word2vec*.

Primeiramente, cada entrada da rede (palavras de um texto) é codificada em um vetor $x_{ck} = \{x_1, x_2, ..., x_V\}$, em que V é o tamanho do vocabulário (conjunto de todas as palavras que aparecem nos textos), c é o identificador da palavra e k, o identificador do texto (como apresentado na Figura 2.2). O item x_i é 1 somente se i corresponde ao identificador da palavra c. Caso contrário, x_i é 0, isto é, apenas uma posição do vetor de cada palavra terá o valor 1 (apenas a que corresponde à palavra). Tal codificação é chamada de *one-hot encoding*. A partir desta entrada, o modelo calcula a média desses vetores e a multiplica pelo vetor de pesos da camada escondida h, gerando a saída y. Detalhes sobre o treinamento da rede podem ser encontrados em [Rong 2014].

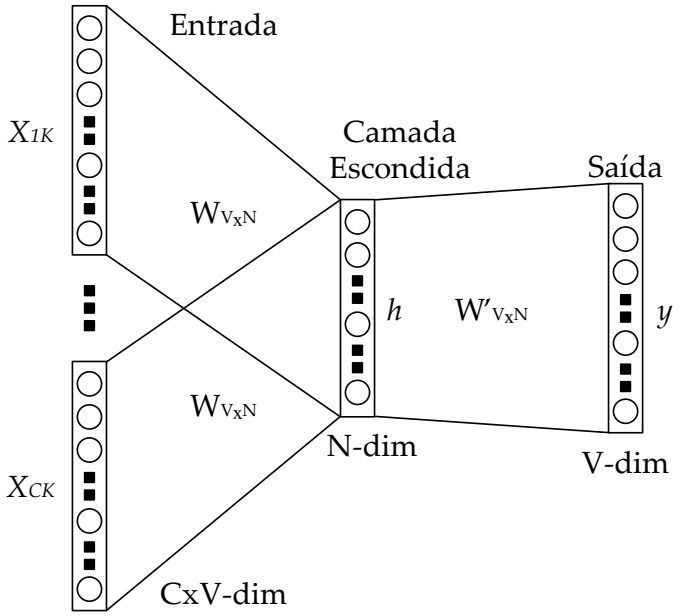


Figura 2.2. Arquitetura *CBOW* do *word2vec*.

O Código 13 apresenta uma forma de se extrair características utilizando o *Word2Vec*. Para isso, implementamos uma classe chamada de `MeanEmbeddingVectorizer`, que utiliza o modelo `Word2Vec`[15] implementado pela biblioteca gensim [Řehůřek and Sojka 2010]. Nesta classe, definimos apenas dois parâmetros (mais parâmetros podem ser adicionados ao modificar essa classe): `size`, que define a dimensão utilizada para projetar as palavras e `min_count`, que define o número mínimo de aparições para que uma palavra seja considerada válida. No método `fit`, treinamos o modelo `Word2Vec` com os dados de treino e obtemos a projeção desses dados. Já no método `transform`, os dados passados como parâmetro utilizam a projeção criada para criar um vetor de características para cada exemplar, com base na média de suas palavras nessa projeção (no caso, caso a palavra não exista, a mesma é considerada como sendo um vetor de zeros). Logo, a classe que implementamos funciona de forma similar a classe implementada para extrair características utilizando o TF-IDF (do *Scikit-Learn*).

2.4.2. Normalização

Um passo importante na extração de características é a normalização já que poucos algoritmos de aprendizado de máquina conseguem ter um bom desempenho quando as características possuem escalas diferentes [Gron 2017]. Alguns algoritmos podem dar mais relevância a valores altos, mesmo que eles representem coisas totalmente diferentes. Por exemplo, se duas características fossem altura, em metros, e peso, em quilos, para classificar o quão saudável uma pessoa é, a diferença de escala entre os dois daria uma significância maior para o peso, já que ele varia em escala muito mais que a altura de uma pessoa. Geralmente utilizam-se dois tipos de normalização [Gron 2017]: *min-max scaling* e *standardization*.

[15] `https://radimrehurek.com/gensim/models/word2vec.html`


```
from gensim.models import Word2Vec
# classe responsavel por treinar modelo w2v e extrair caracteristicas
class MeanEmbeddingVectorizer(object):
    # salva parametros
    def __init__(self, size, min_count=1): #word2vec):
        self.size = size
        self.min_count = 1
    # treina w2v
    def fit(self, X):
        # treina rede neural w2v
        w2v = Word2Vec(X,size=self.size,min_count=self.min_count)
        # obtem projecao das palavras
        self.word2vec = dict(zip(w2v.wv.index2word,w2v.wv.vectors))
        # obtem dimensao dos dados
        self.dim = len(list(word2vec.values())[0])
        return self
    # transforma dados (extrai caracteristicas)
    def transform(self, X):
        # calcula media das palavras de X
        return np.array([
            np.mean([self.word2vec[w] for w in words if w in self.word2vec]
                    or [np.zeros(self.dim)], axis=0)
            for words in X
        ])
... # leitura e divisao treino e teste
# inicializa w2v com maximo de 200 features
m = MeanEmbeddingVectorizer(size=200)
# treina w2v
m.fit(train_texts)
# transforma textos para caracteristicas
train_features = m.transform(train_texts)
test_features = m.transform(test_texts)
# imprime forma das caracteristicas
print(train_features.shape, test_features.shape)
```

```
(25091, 200) (25090, 200)
```

Código 13. Utilizando *Word2Vec* para extrair características de uma lista de bibliotecas (mesma do Código 12) do *dataset Brazilian Malware*.

2.4.2.1. *Min-Max Scaling*

O *MinMax* reescala todas as características em um intervalo entre zero e um, utilizando a Equação 3, em que x_i é o valor da característica x de um exemplar i, $min(x)$ é o menor valor dessa característica no conjunto de treino e $max(x)$, o maior valor dessa característica no mesmo conjunto [Gron 2017].

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3}$$

A biblioteca *scikit-learn* [Pedregosa et al. 2011] possui uma classe chamada `MinMaxScaler`[16] que efetua a normalização, como demonstrado no Código 14. Primeiramente, inicializa-se o *min-max*, especificando o intervalo em que as características devem ser normalizadas (o que é uma vantagem em relação ao *Standardization*, já que alguns algoritmos funcionam melhor com determinados intervalos, como uma rede neural – intervalo de 0 a 1 – e um SVM com *kernel RBF* – intervalo de -1 a 1). Então, o mesmo utiliza o conjunto de treino para extrair os menores e maiores valores para cada característica (através do método `fit`) para então poder normalizá-las (através do método `transform`).

```
from sklearn.preprocessing import MinMaxScaler
... # extracao de caracteristicas
# inicializa minmax
scaler = MinMaxScaler(feature_range=(0, 1))
# treina minmax
scaler.fit(train_features.toarray())
# normaliza caracteristicas
train_features_norm =
  ↪ scaler.transform(train_features.toarray())
test_features_norm =
  ↪ scaler.transform(test_features.toarray())
# imprime forma das caracteristicas
print(train_features_norm.shape,
  ↪ test_features_norm.shape)
```

```
(25091, 200) (25090, 200)
```

Código 14. Utilizando *MinMax* para normalizar características extraídas.

2.4.2.2. *Standardization*

O método *standardization* normaliza os dados subtraindo de cada característica seu valor médio (com base nos dados de treino) e dividindo esse valor pela variância, de modo

[16] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`

que a distribuição resultante tenha variância unitária, como demonstrado na Equação 4, em que x_i é o valor da característica x de um exemplar i, u é a média dos exemplares de treino dessa mesma característica e s, a variância dos exemplare de treino, também dessa característica. Uma vantagem desse método com relação ao *min-max* é que ele não é tão afetado por *outliers* como o *min-max*, já que se houver algum *outlier* nos dados de treino do *min-max*, ele levará em consideração e deixará um intervalo grande entre os demais dados "normais".

$$z_i = \frac{x_i - u}{s} \tag{4}$$

De forma similar ao *min-max*, o *Standardization* também já possui implementação no *scikit-learn* [Pedregosa et al. 2011], implementado na classe `StandardScaler`[17], como demonstrado no Código 15. Basta inicializá-lo, treinar com os dados de treino para obter as médias e variância de cada caraterística (através do método `fit`) e, finalmente, normalizar todos os dados (através do método `transform`).

```
from sklearn.preprocessing import StandardScaler
... # extracao de caracteristicas
# inicializa standardization
scaler = StandardScaler()
# treina standardization
scaler.fit(train_features.toarray())
# normaliza caracteristicas
train_features_norm =
  ↪ scaler.transform(train_features.toarray())
test_features_norm =
  ↪ scaler.transform(test_features.toarray())
# imprime forma das caracteristicas
print(train_features_norm.shape,
  ↪ test_features_norm.shape)
```

```
(25091, 200) (25090, 200)
```

Código 15. Utilizando *Standardization* para normalizar características extraídas.

2.5. Construção de Modelos

Nesta Seção apresentaremos os modelos de aprendizado de máquina mais utilizados nos problemas envolvendo segurança, incluindo suas definições teóricas, algoritmos e códigos para utilizá-los. Dentro os modelos, incluem-se classificadores, detectores e agrupadores, além de um breve comentário sobre as bibliotecas utilizadas para implementá-los.

[17] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

2.5.1. Classificadores

Os classificadores têm como objetivo classificar determinado exemplar de entrada em uma classe previamente conhecida por ele durante o treinamento. O treinamento é a fase em que o classificador aprende os padrões de cada classe com os dados fornecidos (juntamente com seus rótulos), regulando seus parâmetros para o problema em questão. Este tipo de problema é chamado de aprendizado supervisionado [Bishop 2006]. Após o treino, o modelo pode ser utilizado para classificar dados desconhecidos, podendo então ser utilizado. Os classificadores apresentados nessa seção incluem: *K-Nearest Neighbors*, baseado em vizinhança, *Random Forest*, um *ensemble* baseado em árvores de decisão, *Support Vector Machine*, baseado na construção de um hiperplano ótimo, e *Multi-Layer Perceptron*, um tipo de rede neural muito utilizada em *deep learning*.

2.5.1.1. *K-Nearest Neighbors* (KNN)

K-Nearest Neighbors (KNN) é um modelo de aprendizado de máquina baseado em distância, cuja classificação de uma nova instância é baseada na distância dos k exemplares de treino mais próximos. Assim, um novo exemplo desconhecido será classificado como sendo da classe que mais ocorre dentre esses k exemplos, como mostrado na Figura 2.3 [Michie et al. 1994], em que uma nova instância será classificada como vermelha, quando $k = 3$, verde, quando $k = 5$ e desconhecida quando $k = 6$ (o resultado é um empate, por isso números pares não são recomendados para problemas binários).

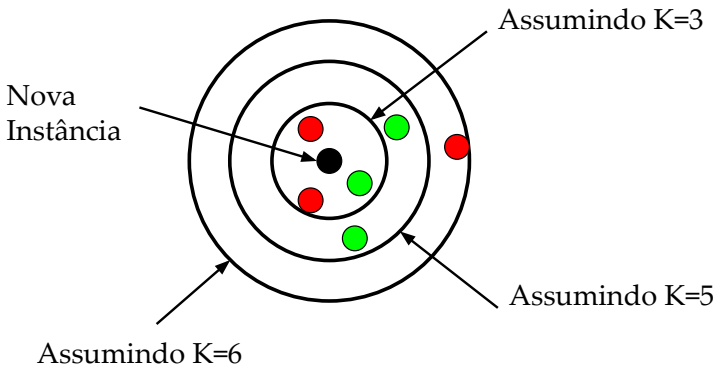


Figura 2.3. Exemplo de funcionamento do KNN.

A distância geralmente utilizada pelo KNN é a distância Euclideana. Dada uma instância x, descrita por (a_1(x),a_2(x),...,a_n(x)), em que a_i(x) é a i-ésima característica, a distância entre duas instâncias x_i and x_j é definida pela Equação 5 [Mellish 2017].

$$d(x_i,x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2)} \tag{5}$$

2.5.1.2. *Random Forest*

Random forest é um classificador que consiste em uma coleção (chamado de *ensemble*) de classificadores baseados em árvores de decisão, cada um treinado usando seleção aleatória de exemplos e características (*boostrap agreegating* ou *bagging*) e tendo como saída um voto para determinada classe, que será utilizada para tomar a decisão final através de uma votação [Breiman 2001]. A Figura 2.4 apresenta um exemplo de *random forest*, contendo L árvores usando *bagging*. Para entender melhor esse ensemble, explicaremos como funciona uma árvore de decisão.

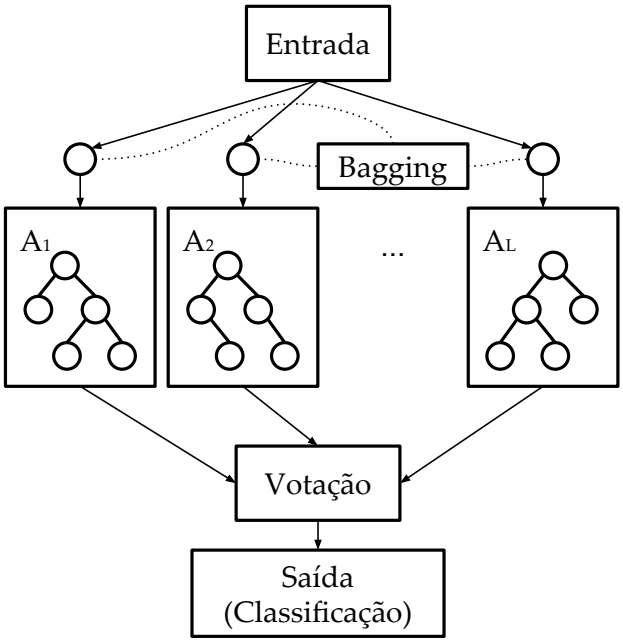


Figura 2.4. Exemplo de *Random Forest*.

Uma árvore de decisão é basicamente um classificador que cria um conjunto de regras *if-then* para classificar novos exemplares. Essas regras melhoram a leitura de resultados por humanos, já que tornam mais fácil de entender e quais características são importantes. A classificação de novas instâncias é feita iniciando-se da raiz da árvore até um nó folha, que provê a classe da instância. Cada nodo interno da árvore representa um teste de alguma características da instância e cada aresta representa possíveis valores que essa característica possui, alterando o caminho de acordo com o mesmo [Mitchell 1997]. A Figura 2.5 mostra um exemplo de árvore de decisão para classificar se um determinado dia é bom para jogar tênis (baseado em condições climáticas).

O treino de uma árvore de decisão é baseado em duas métricas: entropia e ganho de informação. A entropia, representada pela Equação 6, em que S é o conjunto de exemplares de treino e c, o número de classes, mede a homogeneidade dos exemplos.

$$Entropia(S) = \sum_{i=1}^{c} -p_i \log_2(p_i) \tag{6}$$

O ganho de informação, como mostrado na Equação 7, é usado como métrica

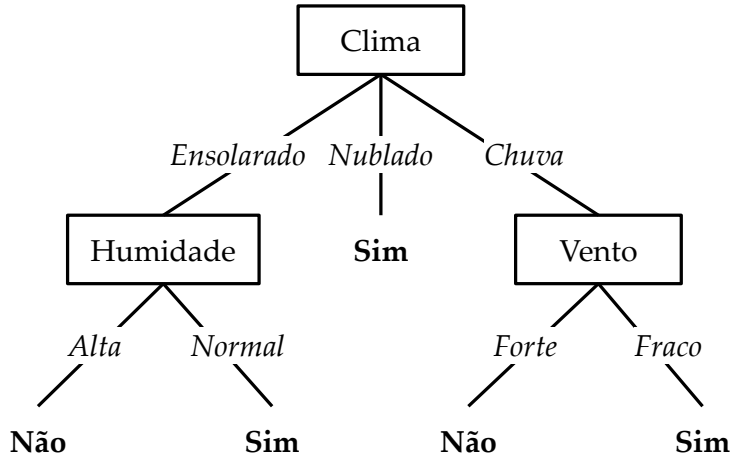


Figura 2.5. Exemplo de árvore de decisão para classificar se um dia é bom para jogar tênis, baseado em condições climáticas.

para seleção de característica. A árvore é construída de acordo com o ganho de informação de uma característica A, já que sempre é utilizado a característica com maior ganho de informação como divisão para criar novas arestas e caminhos para cada valor dessa característica (ou intervalos, no caso de características numéricas) [Mitchell 1997].

$$Ganho(S,A) = Entropia(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Entropia(S_v) \tag{7}$$

O algoritmo para treinar uma árvore de decisão (o algoritmo básico, chamado ID3), dado um conjunto de exemplos (exemplos de treino), seus rótulos (classe a ser detectada pela árvore) e as características, é o seguinte [Mitchell 1997]:

1. Criar nodo raiz para a árvore.
2. Se todos os exemplos são positivos, retornar um único nodo raiz, com rótulo $= +$.
3. Se todos os exemplos são negativos, retornar um único nodo raiz, com rótulo $= -$.
4. Se o conjunto de características é vazio, retorna um único nodo raiz, com o rótulo igual ao valor mais presente nos rótulos.
5. Caso contrário, fazer o seguinte:
   (a) Seja A a característica que melhor classifica os exemplos de treino, isto é, aquela com maior ganho de informação, como já definido na Equação 7.
   (b) A característica escolhida para a raiz é A.
   (c) Para cada valor possível v_i de A:
      i. Adicionar um novo caminho abaixo da raiz, correspondente ao teste $A = v_i$.
      ii. Seja $exemplos_{v_i}$ o subconjunto de exemplos que possuem valor v_i para A:

A. Se $exemplos_{v_i}$ é vazio, então, abaixo desse ramo, adicionar um nó folha com o rótulo igual ao rótulo mais presente no conjunto total de exemplos.

B. Caso contrário, abaixo desse ramo, criar uma nova sub-árvore, voltando ao passo 1 usando um subconjunto de exemplos ($exemplos_{v_i}$) e características ($caracteristicas - \{A\}$).

2.5.1.3. *Support Vector Machine* (SVM)

O *Support Vector Machine* (SVM), originalmente desenvolvido para classificação binária (classificação de apenas duas classes), procura, em problema linearmente separáveis (isto é, quando uma linha separa as classes), a construção de um hiperplano como superfície de decisão (uma fronteira), em que a separação entre os exemplos é máxima. Quando os padrões não são linearmente separáveis, o SVM pode executar uma função de mapeamento que projeta os dados em um espaço em que os dados se tornam linearmente separáveis. A principal ideia desse classificador é maximizar a margem do hiperplano dos dados de treino. Um hiperplano ótimo é aquele cuja distância da margem para a classe positiva é a mesma distância para a classe negativa. A Figura 2.6 ilustra um hiperplano ótimo definido pelos vetores de suporte, os dados de treino mais próximos a ele [Cortes and Vapnik 1995, Fradkin and Muchnik 2006].

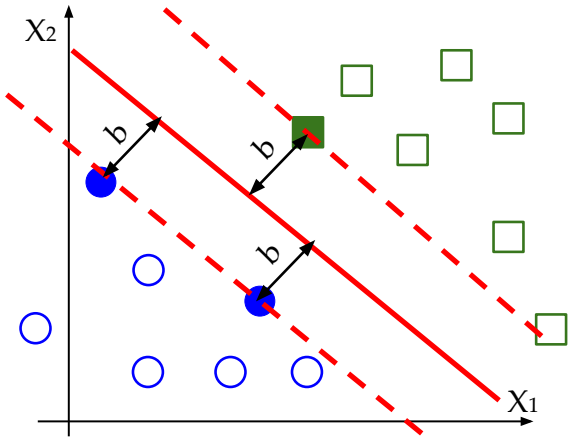


Figura 2.6. Exemplo de hiperplano ótimo de um SVM.

Na maioria das vezes, os problemas não são linearmente separáveis. Por conta disso, é necessário projetar os dados em um espaço em que eles se tornem linearmente separados, chamado de *feature space*. A função de *kernel* é responsável por essa projeção e esse processo é chamado de *kernel trick*. Depois de projetados, é possível encontrar um hiperplano que separa os dados nesse novo espaço. A Figura 2.7 exemplifica o uso do *kernel trick* para projetar os dados em outra dimensão.

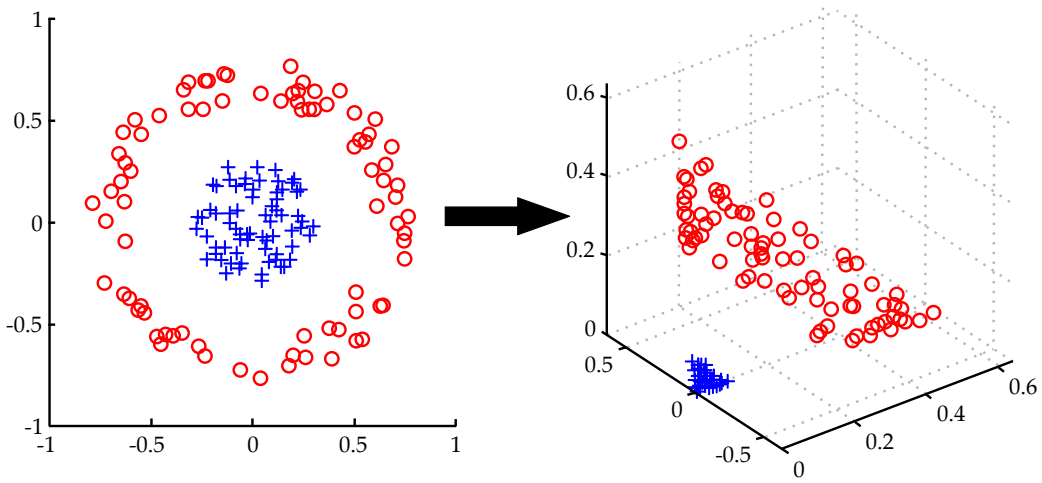


Figura 2.7. Exemplo de projeção de um problema não-linear utilizando *kernel trick*, em que os dados são projetados em outra dimensão (bi-dimensional para tri-dimensional) [Jordan 2017].

A função de decisão responsável por construir o hiperplano é definida pela Equação 8, em que K é a função de *kernel*, α e b são parâmetros encontrados durante o treinamento, x_i são os vetores de características e y_i, seus rótulos.

$$f(x) = \sum_i \alpha_i y_i K(x, x_i) + b \tag{8}$$

Como a maioria dos problemas reais envolvem mais de duas classes e o SVM é um classificador binário, é necessário usar uma abordagem de decisão diferente. A maior comum é a abordagem "um contra todos", também chamada de "um contra o resto" [Manning et al. 2008b]. Nesta abordagem, existem q classificadores (SVMs), em que q é o número de classes. Cada SVM c_i é treinado para a classe i, usando como contra-exemplo os exemplares das outras classes. A decisão final pode ser feita através de uma "contagem de votos" [Milgram et al. 2006].

2.5.1.4. *Multi-Layer Perceptron* (MLP)

Uma rede neural é um modelo computacional de um cérebro humano que pode ser composto de centenas ou milhares de neurônios (unidades de processamento). Em geral, é uma máquina que é desenvolvida para modelar a forma com qual o cérebro realiza uma tarefa em particular. O menor componente de uma rede neural é um *perceptron*, uma unidade de processamento que simula um neurônio [Haykin 2009]. A Figura 2.8 mostra a estrutura de um *perceptron*, em que $x = \{x_1, x_2, ..., x_n\}$ são os sinais (entrada), $w = \{w_1, w_2, ..., w_n\}$ são os pesos, $\varphi(.)$ é a função de ativação e y é a saída.

A Equação 9 apresenta a saída (y) de um *perceptron*. Os sinais de entrada são multiplicados pelos pesos, que são adaptados a cada iteração durante o treino, usando uma regra de correção de erro (algoritmo de convergência do *perceptron*). Ele também

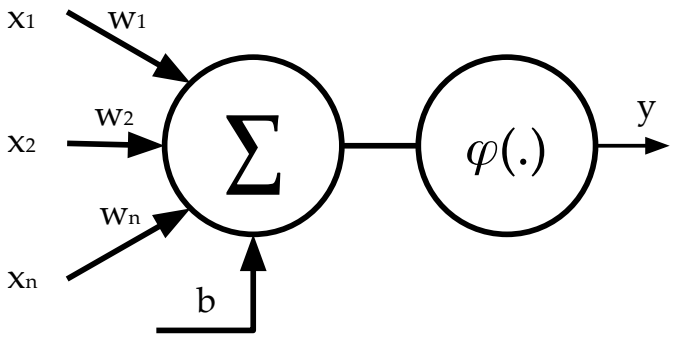


Figura 2.8. Estrutura de um *perceptron*.

inclui um *bias* (b), que tem o efeito de aumentar ou diminuir o sinal da função de ativação. A função de ativação é responsável por determinar a forma e intensidade dos valores transmitidos de um neurônio para outro (ou para a saída), limitando a amplitude da saída [Haykin 2009].

$$y = \varphi(\sum_{i=1}^{n} w_i \times x_i + b) \tag{9}$$

O treino de um único *perceptron* ajuda a entender melhor a operação de uma rede neural. O algoritmo é o seguinte [Haykin 2009]:

1. Inicializar pesos e *bias* com valores aleatórios pequenos.

2. Aplicar padrão de entrada do exemplo corrente e checar a saída da rede (y).

3. Calcular o erro da saída (e), comparando o mesmo (saída y) com o valor esperado (t_j), como mostrado na Equação 10.

$$e = t_j - y \tag{10}$$

4. Se o erro for igual a zero ($e = 0$), significa que a saída está correta. Neste caso, um novo exemplo é apresentado, voltando ao passo 2.

5. Caso contrário, se o erro for diferente de zero ($e \neq 0$), é necessário atualizar os pesos e *bias*, como mostrado nas Equações 11 e 12, respectivamente.

$$w_j = w'_j + e \times x_j \tag{11}$$

$$b = b' + e \tag{12}$$

6. Voltar ao passo 2 e apresentar um novo exemplo a rede. O critério de parada pode ser baseado em número de iterações, erro médio, acurácia, etc.

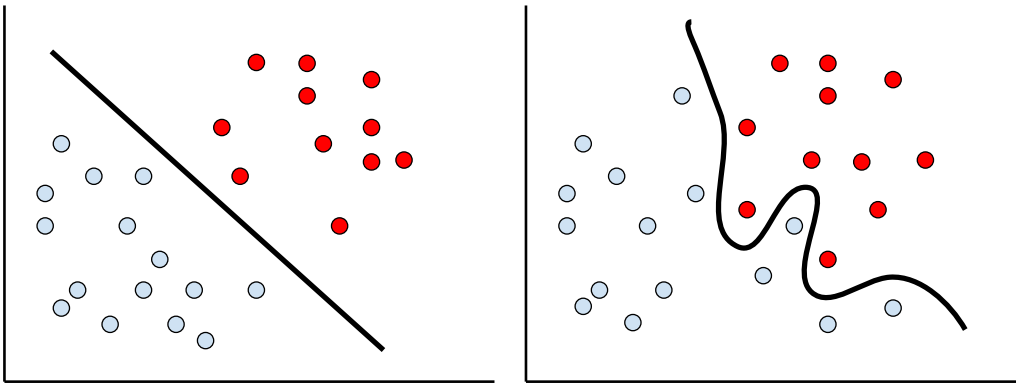


Figura 2.9. Problema linear (esquerda) contra não problema não-linear (direita).

Apesar de ser eficiente, um único *perceptron* pode apenas resolver problemas linearmente separáveis, o que, na maioria das vezes, não acontece no mundo real. Por conta disso, redes neurais geralmente combinam mais que um neurônio, o que torna possível separar problemas não-lineares [Haykin 2009]. A Figura 2.9 mostra dois exemplos de problemas, o primeiro (esquerda), linear e o segundo (direita), não-linear. Como a fronteira de decisão do *perceptron* é definido por uma linha, que resolve problemas como o primeiro caso. Uma rede mais complexa, tal como a *multilayer percetron*, composta por vários neurônios, pode resolver o segundo problema.

Um rede neural do tipo *multilayer perceptron* (MLP) é composta por nós de entrada, uma camada ou mais de *perceptrons*, chamada de camadas escondidas, e os neurônios de saída. Todas as camadas, exceto a entrada, são compostas por neurônios, cada uma delas inicializadas por pesos e *bias* aleatórios, como um *perceptron* qualquer. Esse tipo de rede é progressiva, i.e., os neurônios de uma determinada camada são apenas conectadas com a próxima camada. Assim, a entrada passa por todas as camadas existentes. O número de nós de entrada é a dimensionalidade dos dados de entrada (número de características) e o número de neurônios da saída é geralmente composta pelo número de classes do problema (neste caso, cada neurônio representa uma classe, i.e., o valor de saída do neurônio é diretamente relacionada a sua respectiva classe. Quanto maior o valor, maior as chances de ser daquela classe) [Haykin 2009, Bishop 2006]. A Figura 2.10 apresenta uma rede *multilayer perceptron* com duas camadas escondidas e três neurônios de saída.

Um algoritmo muito importante para redes neurais em geral é o *back propagation*, um método computacional eficiente para o treinamento de múltiplos *perceptrons*. Brevemente, esse algoritmo implementa o grandiente descendente, computando derivadas parciais de uma função aproximada $F(w,x)$ calculada pela rede, ajustando os pesos de acordo com a entrada [Haykin 2009].

2.5.2. Detectores

A detecção de mudança é um tema fundamental para muitos problemas da área de segurança, uma vez que a distribuição dos dados geralmente está em constante mudança para evadir sistemas de segurança. Essas mudanças são chamadas de *Concept drift*, si-

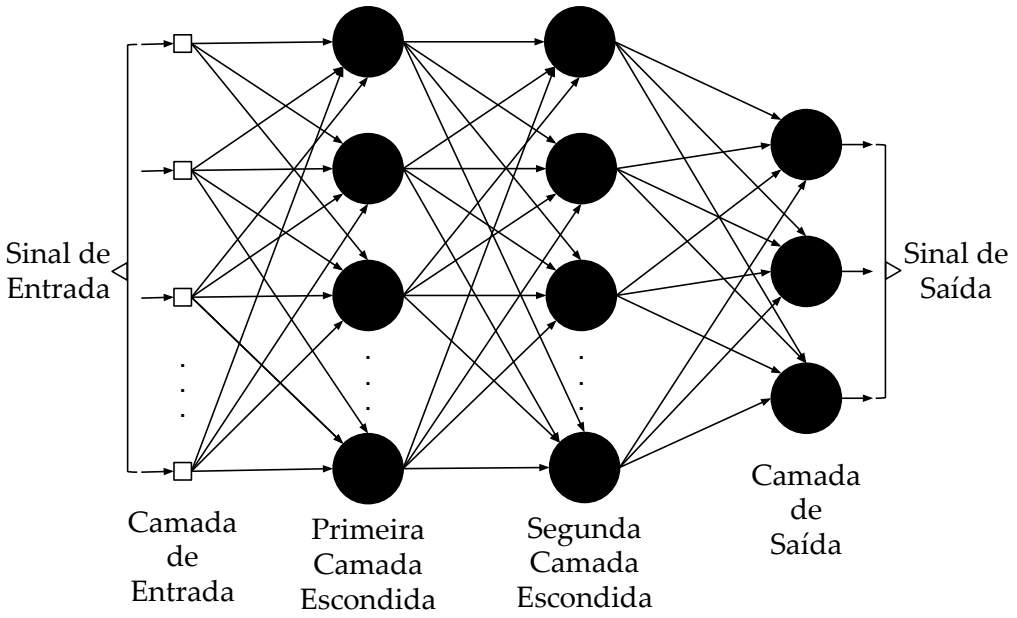
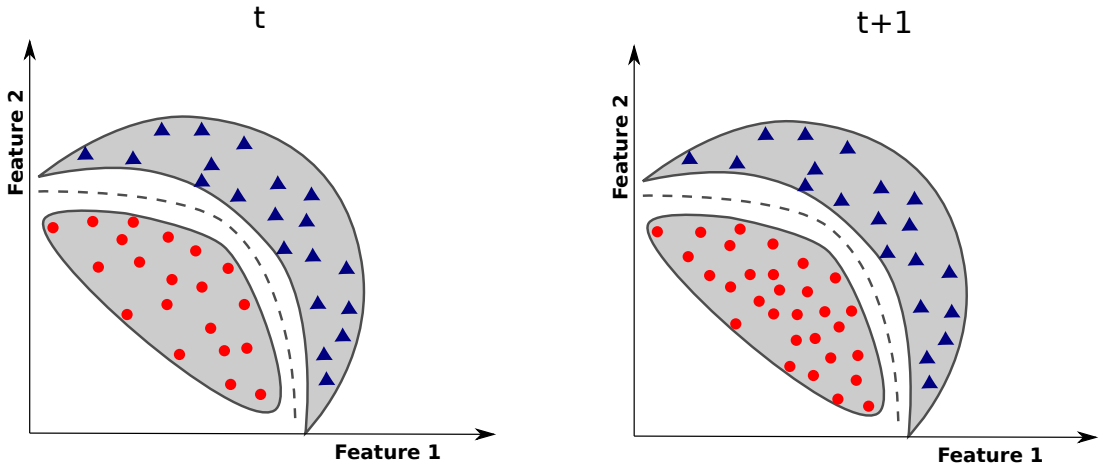


Figura 2.10. Exemplo de rede neural *multilayer perceptron* (MLP).

tuação em que a relação entre os dados de entrada e sua classe mudam com o passar do tempo. Isso geralmente acontece quando há mudanças em um contexto escondido, se tornando complicado de lidar, já que este problema acontece em diferentes campos de pesquisa. Cada exemplo dos dados de entrada é representado por um vetor de características $x = [x_1, x_2, ..., x_L]$, em que L é o número de características que são usadas para determinar sua classe y, de acordo com uma probabilidade *a posteriori* $P(y,x)$. $P(x)$ é definido como distribuição de características e $P(y)$ como probabilidades *a priori*. Na literatura, existem dois tipos de *concept drift*: virtual e real [Wang et al. 2011, Gama et al. 2014].

O *concept drift* virtual acontece quando a distribuição dos dados muda, i.e., $p(x)$ muda, sem alterar $p(y,x)$. A Figura 2.11 mostra um exemplo de *concept drift* virtual com mudanças na distribuição dos dados no tempo t e $t+1$, em que a classe vermelha é mais prevalente mas não leva a mudanças na melhor fronteira de decisão entre as classes [Almeida et al. 2015].

Para ilustrar o problema causado pelo *concept drift* virtual, a Figura 2.12 apresenta a distribuição de uma característica para as classes azul e vermelha (apresentado na Figura 2.11) nos tempos t e $t+1$. A linha vertical pontilhada em ambas as Figuras é o limiar (*threshold*) que separa ambas as classes (o custo de classificação errônea da classe vermelha é maior que da classe azul), que são igualmente distribuídas no tempo t (Figura 2.12(a)), diferente do tempo $t+1$ (Figura 2.12(b)), que possui maior presença da classe vermelha. Apesar do fato da média e desvio padrão de ambas as classes permanecer a mesma, com o *threshold* na mesma posição, i.e., mantendo o classificador sem nenhuma mudança, a probabilidade de encontrar um objeto vermelho como sendo azul aumenta, como mostrado na área vermelha escura na Figura 2.12(b) [Almeida et al. 2015].



(a) Projeção de exemplos no tempo t.

(b) Projeção de exemplos no tempo $t+1$.

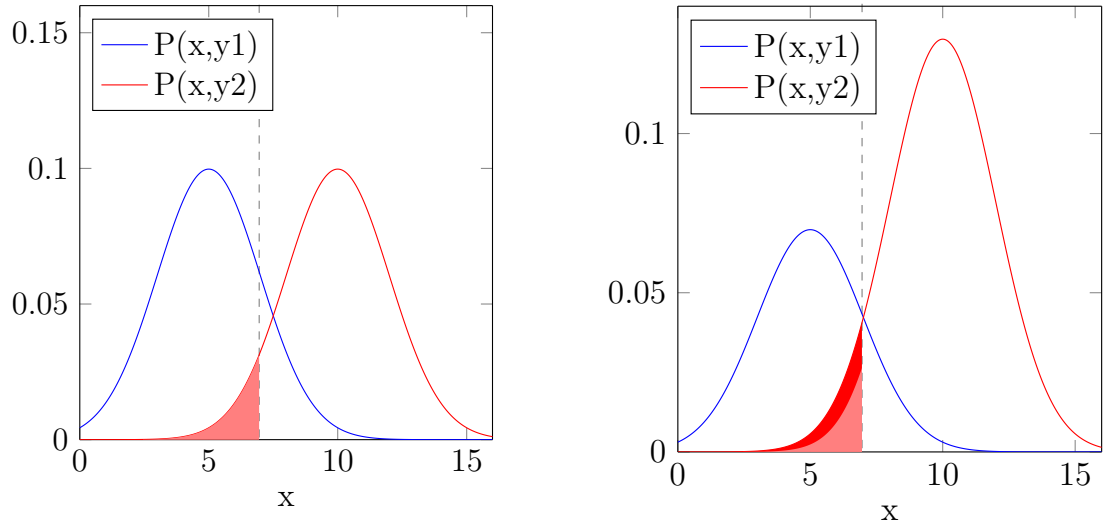
Figura 2.11. Exemplo de *concept drift* virtual em que $P_t(y) \neq P_{t+1}(y)$ [Almeida et al. 2015].

Já o *concept drift* real acontece quando há mudança em $P(y,x)$ com ou sem mudança em $p(x)$, i.e., a relação entre as classes e vetores de características muda com o passar do tempo. Um exemplo clássico é relacionado ao problema de *spam* de e-mails, em que um e-mail representado por um vetor x_e pode ser considerado como spam em determinado tempo t e pode não ser no tempo $t+1$, devido a mudanças de comportamento dos usuários. Como um exemplo de *concept drift* real, a Figura 2.13 mostra um problema de duas classes com um *drift* nas probabilidades *a posteriori*, causando mudanças nas fronteiras do tempo t e $t+1$ (Figuras 2.13(a) e 2.13(b),respectivamente) e forçando uma atualização do classificador [Almeida et al. 2015].

Para ambos os tipos de *concept drift*, existem diversos detectores, cada um deles utiliza uma estratégia diferente para analisar as mudanças ocorridas com o passar do tempo. Neste curso, selecionamos três detectores que são comumente utilizados na literatura: DDM (*Drift Detection Method* [Gama et al. 2004]), EDDM (*Early Drift Detection Method* [Baena-Garćia et al. 2006]) and ADWIN (*ADaptive WINdowing* [Bifet and Gavaldà 2007]).

2.5.2.1. *Drift Detection Method* (DDM)

A ideia geral do *Drift Detection Method* (DDM) é monitorar a taxa de erro de um modelo, enquanto os exemplos são apresentados ao classificador em sequência, um *data stream* [Gama et al. 2004]. Geralmente, quando a distribuição dos dados é estacionária, o erro do modelo deve cair ou manter-se estável conforme mais dados são utilizados. Quando a taxa de erro aumenta, o DDM utiliza isso como evidência para detectar uma mudança de conceito [Albert Bifet 2018]. Seja p_t a taxa de erro do classificador e $s_t = \sqrt{\frac{p_t(1-p_t)}{t}}$ o desvio padrão, ambos no tempo t, o DDM salva a menor taxa de erro p_{min} e menor desvio padrão s_{min} observados até o tempo t e então realiza as seguintes



(a) Distribuição de classes no tempo t.

(b) Distribuição de classes no tempo $t+1$.

Figura 2.12. Exemplo de *concept drift* virtual em que $P_t(y) \neq P_{t+1}(y)$ [Almeida et al. 2015].

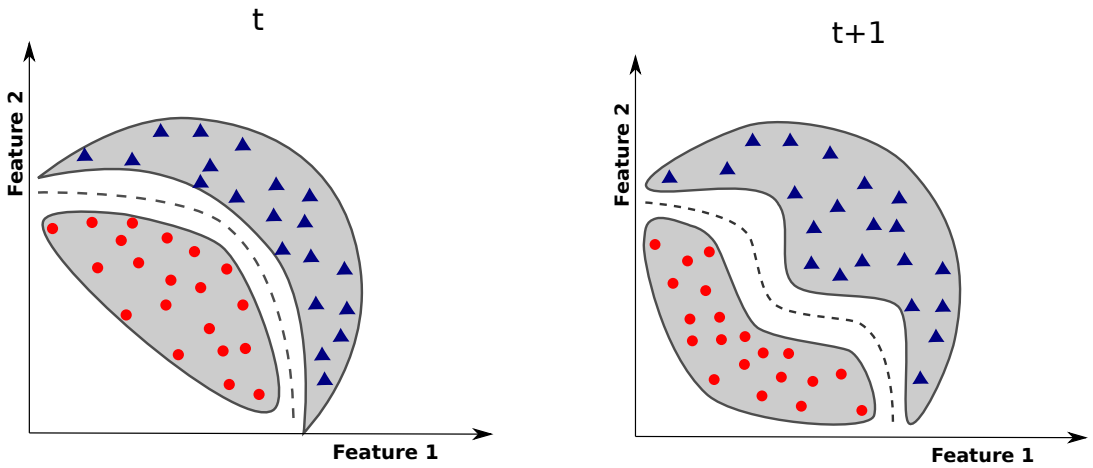
verificações [Albert Bifet 2018]:

- Se $p_t + st \geq p_{min} + 2 * s_{min}$, considera-se que o modelo está em estágio de *warning*, sugerindo que um *drif* está começando. A partir desse ponto, todos os exemplos são armazenados em um *buffer*, já que uma mudança pode estar prestes a acontecer.
- Se $p_t + st \geq p_{min} + 3 * s_{min}$, considera-se que uma mudança aconteceu. O modelo utilizado é descartado e um novo é construído com os exemplos armazenados no *buffer* desde que o *warning* ocorreu. Os valores de p_{min} e s_{min} são resetados.

Embora seja genérico e simples de se utilizar, o DDM pode ser muito lento para responder a mudanças em alguns casos, já que muitos exemplos podem ser observados até que o nível de *drift* seja efetivamente ativado, podendo até armazenar muitos exemplos em memória, deixando o processo custoso [Albert Bifet 2018].

2.5.2.2. *Early Drift Detection Method* (EDDM)

O *Early Drift Detection Method* (EDDM) funciona de forma similar ao DDM. Entretanto, em vez de considerar a taxa de erros, o EDDM considera a distância entre dois erros. Segundo os autores do método, conforme um modelo vai sendo treinado (com novos exemplares), ele melhora suas predições e a distância entre dois erros aumenta. Seja p_i a distância média entre dois erros e s_i seu desvio padrão, o EDDM armazena ambos os valores quando $p_i + 2 * s_i$ atinge seu maior valor (obtendo p_{max} e s_{max}), ou seja, $p_{max} + 2 * s_{max}$ é o ponto em que a distribuição da distância entre erros é máxima [Baena-Garćıa et al. 2006]. Conforme o modelo é atualizado e testado com novas amostras, o EDDM realiza as seguintes verificações, dados dois parâmetros α e β:



(a) Fronteira de decisão no tempo t. (b) Fronteira de decisão no tempo $t+1$.

Figura 2.13. Exemplo de *concept drift* real [Almeida et al. 2015].

- Se $\frac{p_i+2*s_i}{p_{max}+2*s_{max}} < \alpha$, considera-se que o modelo está em estágio de *warning*, armazenando os exemplos vistos a partir desse ponto em um *buffer* e sugerindo o início de possível mudança.
- Se $\frac{p_i+2*s_i}{p_{max}+2*s_{max}} < \beta$, considera-se que uma mudança aconteceu, descartando o modelo utilizado e construindo um novo usando os exemplos armazenados no *buffer* desde que o *warning* ocorreu. Os valores de p_{max} e s_{max} são resetados.

No artigo em que o EDDM foi proposto [Baena-Garćıa et al. 2006], o EDDM foi melhor em todos os experimentos em relação ao DDM, mesmo em *datasets* com grande quantidade de ruídos. Entretanto, o problema relacionado ao uso de memória continua o mesmo do DDM, já que ao entrar em estágio de *warning*, o modelo armazena todos os exemplares até que um *drift* ocorra.

2.5.2.3. *ADaptive WINdowing* (ADWIN)

O detector *ADaptive WINdowing* (ADWIN) age de forma diferente do DDM e EDDM, mantendo estatísticas de janelas deslizantes de tamanhos variáveis, que são utilizadas para computar a média da mudança observada "cortando" essas janelas em diferentes pontos. A propriedade dessas janelas deve ser estatisticamente consistente com a hipótese "não houve mudança no valor médio dentro da janela", i.e., uma parte antiga da janela é jogada fora apenas se existir evidência suficiente de que o valor médio difere do resto da janela [Bifet and Gavaldà 2007]. O ADWIN possui como parâmetro um valor de confiança $\delta \in (0,1)$ e um teste $T(W_0, W_1, \delta)$ que compara a média de duas janelas W_0 e W_1, que decide se ambas pertencem a mesma distribuição, com base nas seguintes verificações [Albert Bifet 2018]:

- Se W_0 e W_1 pertencem a mesma distribuição, então, com probabilidade de pelo menos $1-\delta$, o teste responde que não houve mudança.

- Se W_0 e W_1 são de duas distribuições diferentes cuja média difere por uma quantidade maior que $\varepsilon(W_0, W_1, \delta)$, então, com probabilidade de pelo menos $1-\delta$, o teste responde que houve mudança.

Diferente do DDM e EDDM, o ADWIN pode ser mais eficiente em relação ao uso de memória, já que não é necessário criar um *buffer* para salvar os exemplares, basta utilizar os tamanhos da janelas "cortadas" ao detectar uma mudança.

2.5.3. Agrupadores

Muitas vezes os dados são disponibilizados sem nenhum rótulo, dificultando o uso de classificadores convencionais. Para esses tipos de problemas, utiliza-se o aprendizado não supervisionado [Bishop 2006], normalmente utilizando algoritmos agrupadores (*clustering*), cujo objetivo é agrupar padrões parecidos em grupos que podem ou não ser conhecidos, dependendo do domínio do problema. Neste curso, apresentaremos dois algoritmos muito utilizados para agrupamento de dados: *K-Means* e *DBScan*.

2.5.3.1. K-Means

O algoritmo *K-Means* consiste basicamente em agrupar um conjunto de dados em K grupos, que são definidos com base na média de seus dados – um determinado exemplo pertence ao grupo que está mais próximo a ele [Michie et al. 1994]. O número de grupos K é definido pelo usuário do algoritmo e corresponde ao número de classes gerados após a execução do mesmo. O algoritmo do *K-Means* funciona da seguinte forma, dado um número de grupos K [Rogers and Girolami 2011]:

1. Escolher um número K de grupos. Para cada grupo, um valor aleatório é escolhido no início, considerado o centroide de cada um deles.
2. Para cada exemplar do *dataset*, atribui-se um dado exemplar ao grupo que está mais próximo a ele. .
3. Após rotular todos os dados, um novo centroide é calculado para cada grupo (através da média dos exemplares de cada um).
4. Se algum critério de parada não for atingido (número de iterações, variação média de cada centroide, etc), voltar ao passo 2 até que o mesmo seja atingido.
5. Ao final do algoritmo, serão criados K grupos diferentes para o conjunto de dados.

2.5.3.2. *Density-Based Spatial Clustering of Applications with Noise* (*DBScan*)

Density-based spatial clustering of applications with noise (*DBSCAN*) é um algoritmo que agrupa pontos que estão próximos um dos outros baseado em uma medida de distância (geralmente distância Euclidiana) e um número mínimo de pontos, marcando como *outliers* pontos que estão em regiões de baixa densidade. O *DBSCAN* é composto basicamente por dois parâmetros: *eps* (ε), que especifica o quão perto os pontos devem estar

um dos outros para ser considerado parte de um grupo, isto é, se a distância entre dois pontos for menor ou igual a esse valor, esses pontos são considerados vizinhos (do mesmo grupo); e *minPoints*, o número mínimo de pontos para formar uma região densa, isto é, são necessários pelo menos *minPoints* para que uma região seja considerada densa. Em geral, valores menores de *eps* (ε) são preferíveis para vários *datasets*,e quanto maior o *dataset*, maior o valor de *minPoints* a ser escolhido [Ester et al. 1996].

O *DBSCAN* funciona basicamente da seguinte forma [Schubert et al. 2017]:

1. Encontrar os pontos que estão a uma distância *eps* (ε) de cada ponto e encontrar pontos centrais com mais de *minPoints* vizinhos.

2. Encontrar todos os componentes conectados aos pontos centrais no grafo de vizinhança, ignorando todos os pontos centrais.

3. Atribuir cada ponto não-central ao *cluster* mais próximo se o mesmo está a uma distância *eps* (ε), caso contrário, considera-se que o ponto é um ruído.

Uma das vantagens do *DBSCAN* em relação ao *K-Means* é que o mesmo não precisa definir um número de *clusters* (grupos) a *priori* e o mesmo pode encontrar agrupamentos de diversas formas diferentes [Ester et al. 1996].

2.5.4. Bibliotecas

Neste curso apresentaremos três bibliotecas em que os algoritmos apresentados podem ser utilizados: *Scikit-Learn* [Pedregosa et al. 2011], *Scikit-Multiflow* [Montiel et al. 2018] e *Keras* [Chollet et al. 2015]. Todas elas são *open source*, possuem uma comunidade ativa e são complementares na grande maioria das suas funções. Nesta Subseção, focaremos em como utilizar os classificadores de cada biblioteca mencionada, deixando de lado apenas a avaliação, que será discutida depois, na Seção 2.6. Considera-se que os dados utilizados em cada código apresentado já tiveram suas características extraídas e normalizadas, além de estarem separados em dois conjuntos, treino (`train_features_norm`) e teste (`test_features_norm`), como demonstrado nos Códigos 14 e 15.

2.5.4.1. *Scikit-Learn*

A *Scikit-Learn*[18] [Pedregosa et al. 2011] é uma das bibliotecas mais completas de aprendizado de máquina, com diversos algoritmos já implementados, tanto de classificação, pré-processamento, extração de características e normalização. Por conta disso, essa biblioteca já foi utilizada no curso na parte de extração de características (Seção 2.4), pois implementa o TF-IDF e os dois métodos de normalização apresentados (*Min-Max* e *Standardization*). De todos os algoritmos apresentados, essa biblioteca implementa os

[18] `https://scikit-learn.org/`

classificadores *KNN*[19], *Random Forest*[20] e *SVM*[21], além dos agrupadores *K-Means*[22] e *DBScan*[23]. Vale ressaltar que a maioria dos algoritmos implementados no *Scikit-Learn* são feitos para dados em *batches*, em que são usados apenas lotes coletados durante um tempo determinado (esse tempo geralmente é ignorado, permitindo um "acesso aleatório" dos dados e geralmente assumindo que os dados possuem distribuição estacionária, isto é, que não mudam conforme o tempo passa), sem se preocupar com novos dados a serem coletados (neste caso, caso haja novos lotes de dados, seria necessário treinar um novo modelo do zero para que o mesmo possa aprender com os mesmos).

O uso dos classificadores citados funciona da mesma forma nesta biblioteca, conforme mostrado no Código 16, no qual utilizamos o *Random Forest* e definimos o número de árvores utilizadas como sendo 100 (parâmetro `n_estimators`). Já o Código 17 utiliza o *K-Means* para agrupar os exemplares em dois grupos (parâmetro `n_clusters`), sem se preocupar com os rótulos, já que se trata de aprendizado não-supervisionado. O método `fit_predict` treina o agrupar e já retorna as predições para o conjunto passado como parâmetro (funciona da mesma forma para o *DBScan*). Cada classificador e agrupador citado possui os seus parâmetros (a maioria já explicada nesta mesma Seção), todos eles documentados nas páginas de documentação da biblioteca, já mencionadas.

```
from sklearn.ensemble import RandomForestClassifier
... # extracao de caracteristicas e normalizacao
# inicializa classificador com seus parametros
clf = RandomForestClassifier(n_estimators=100)
# treina classificador
clf.fit(train_features_norm, train_label)
# prediz as classes de teste
test_pred = clf.predict(test_features_norm)
# imprime forma da predicao e do vetor de rotulos
print(test_pred.shape, test_label.shape)
```

```
(25090,) (25090,)
```

Código 16. Utilizando o classificador *Random Forest* para classificar um conjunto de testes criado a partir do *dataset Brazilian Malware*.

[19] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[20] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[21] https://scikit-learn.org/stable/modules/svm.html

[22] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[23] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html


```
from sklearn.cluster import KMeans
... # extracao de caracteristicas e normalizacao
# inicializa kmeans com seus parametros
clustering = KMeans(n_clusters=2)
# treina kmeans (note que os rotulos nao sao utilizados)
# e prediz suas classes usando os grupos criados
train_pred = clustering.fit_predict(train_features_norm)
# imprime forma da predicao e do vetor de rotulos
print(train_pred.shape, train_label.shape)
```

```
(25091,) (25091,)
```

Código 17. Utilizando o agrupador *K-Means* para agrupar conjunto de dados criado a partir do *dataset Brazilian Malware*.

2.5.4.2. *Scikit-Multiflow*

O *Scikit-Multiflow*[24] é uma biblioteca de aprendizado de máquina que segue a "filosofia *Scikits*" focada em problemas para *data streams*. Diferente do formato em *batches*, *data streams* são dados acessados de forma contínua (e sequencial), geralmente em tempo real (o que permite com que sejam utilizado mais facilmente no mundo real), isto é, a qualquer momento um novo exemplar pode chegar, influenciando diretamente o modelo, uma vez que a distribuição normalmente é não-estacionária e possui mudanças ao longo do tempo [Albert Bifet 2018]. A Figura 2.14 compara os dados em *batch* e em *stream*: ao utilizar *batch*, as fases de treino e teste são bem definidas e o conjunto de dados é finito, diferente de uma *stream*, em que os dados podem fazer parte do conjunto de treino e teste ao longo do tempo (dependendo de mudanças de conceito e evolução dos dados), que é, teoricamente, infinito, uma vez que novos dados podem chegar a qualquer momento. Dos classificadores apresentados nesta Seção, a biblioteca *scikit-multiflow* implementa algumas adaptações (com algumas mudanças para problemas envolvendo *data streams*) do *KNN*[25] (esta versão já inclui um detector de mudanças ADWIN embutido no classificador) e *Random Forest*[26] (por padrão, já possui o detector de mudanças ADWIN embutido – é possível alterá-lo e até mesmo desativá-lo). Dos detectores de mudança citados, todos eles são implementados: DDM[27], EDDM[28] e ADWIN[29]. Até o momento deste trabalho, nenhum agrupador citado foi implementado pela biblioteca.

[24] https://scikit-multiflow.github.io/

[25] https://scikit-multiflow.github.io/scikit-multiflow/_autosummary/skmultiflow.lazy.KNNAdwin.html

[26] https://scikit-multiflow.github.io/scikit-multiflow/_autosummary/skmultiflow.meta.AdaptiveRandomForest.html

[27] https://scikit-multiflow.github.io/scikit-multiflow/_autosummary/skmultiflow.drift_detection.DDM.html

[28] https://scikit-multiflow.github.io/scikit-multiflow/_autosummary/skmultiflow.drift_detection.EDDM.html

[29] https://scikit-multiflow.github.io/scikit-multiflow/_autosummary/skmultiflow.drift_detection.ADWIN.html

(a) Dados em *Batch*.

(b) Dados em *Stream*.

Figura 2.14. Comparação: *Batch* VS *Stream*.

O Código 18 apresenta um exemplo de como utilizar um modelo da biblioteca, no caso o `AdaptiveRandomForest`, inicializado com 10 árvores e detector de *drift* ADWIN. A principal diferença para o *Scikit-Learn* é que o método para treino se chama `partial_fit`, que implementa um "treino parcial" e permite que o modelo seja atualizado com novos dados. Além disso, utilizamos a classe `DataStream` para criar uma *stream* com os dados de teste. Então, para cada dado da *stream*, o modelo prediz sua classe e atualiza o modelo com o novo dado (neste mesmo momento, como o classificador possui um detector de *drift* embutido, já é avaliado se houve mudança de conceito nos dados para atualizar o modelo adequadamente – descartando dados antigos se for preciso e mantendo os mais recentes, desde o último *warning*, como já explicado). Nesta implementação, usamos um vetor para salvar todas as predições feitas pelo modelo ao longo da *stream*. Todos os demais classificadores e detectores de *drift* dessa implementação estão documentados nas páginas de documentação da biblioteca, também já mencionadas.

Para utilizar um detector de *drift* em qualquer classificador, pode-se seguir os exemplos dos Códigos 19 e 20. O primeiro deles (Código 19), inicializa um classificador, no caso, uma árvore de decisão *HoeffdingTree*, a mesma utilizada pelo *Random Forest*, e o treinamos com os dados de treino. Um segundo classificador, com a mesma árvore, é criado para ser treinada apenas quando o primeiro modelo entrar em estágio de *warning*. Em seguida, o detector de drift (no caso, o DDM) é inicializado, bem como a *stream* contendo os dados de teste. No Código seguinte (Código 20), cria-se um vetor para armazenar as predições, um contador para contar o número de dados já processados na *stream* e um vetor para armazenar os pontos em que houveram *drift*. Ao iterar sobre a *stream*, obtém-se os dados do próximo exemplar, incrementa-se o contador de exemplares e faz-se a predição do exemplar obtido no classificador corrente. Essa predição é então comparada ao valor real do exemplar pelo detector de *drift*, que checa se está em fase de *warning* ou se realmente houve um *drift*. Se um *warning* é detectado, o classificador dois é habilitado, apresentando o exemplar corrente para ele. Caso um *drift* seja detectado, o ponto de *drift* é salvo e o classificador secundário assume o lugar do primeiro, criando um novo classificador para a próxima fase de *warning* e resetando o detector. De qualquer forma, o classificador principal é atualizado com o novo exemplar. Ao final, os pontos de *drift* são impressos. A parte interessante dessa implementação é que, independente do problema a ser resolvido, a solução não fica presa ao classificador utilizado, já que o detector não é embutido no mesmo (e mesmo no caso do `AdaptiveRandomForest`, é possível desabilitar o detector interno dele), permitindo com que você desenvolva sua própria lógica ao detectar uma mudança de conceito.

2.5.4.3. *Keras*

A biblioteca *Keras* é uma API de redes neurais de alto nível implementada em Python, que permite que seja capaz utilizar bibliotecas de mais baixo nível, tais como *TensorFlow* [Abadi et al. 2015], *CNTK* [Seide and Agarwal 2016] e *Theano*, agilizando o processo de desenvolvimento de pesquisas utilizando redes neurais [Theano Development Team 2016] [Chollet et al. 2015][30]. Por se tratar de uma biblioteca específica de redes neurais, dos modelos citados ela implementa apenas o *Multi-Layer Perceptron*, sendo possível uma vasta variedade de estruturas de rede neurais diferentes.

[30] https://keras.io/

```
from skmultiflow.meta import AdaptiveRandomForest
from skmultiflow.data import DataStream
# inicializa classificador
clf = AdaptiveRandomForest(n_estimators=10)
# treina classificador
clf.partial_fit(train_features_norm, train_label)
# cria stream com dados de teste
s = DataStream(test_features_norm, test_label)
# prepara stream para uso
s.prepare_for_use()
# cria vetor final de predicao
test_pred = []
# itera sobre stream
while s.has_more_samples():
    # obtem caracteristicas e rotulo do proximo exemplar
    sample_features, sample_label = s.next_sample(1)
    # prediz exemplar
    sample_pred = clf.predict(sample_features)
    # adiciona predicao ao vetor de predicao
    for l in sample_pred:
        test_pred.append(l)
    # atualiza modelo com novo exemplar
    clf.partial_fit(sample_features, sample_label)
# transforma vetor de predicao em vetor numpy
test_pred = np.array(test_pred)
# imprime forma da predicao e do vetor de rotulos
print(test_pred.shape, test_label.shape)
```

```
(25090,) (25090,)
```

Código 18. Utilizando o classificador *Adaptive Random Forest* (com detector de *drift* ADWIN) para classificar um conjunto de testes criado a partir do *dataset Brazilian Malware*.


```
from skmultiflow.trees import HoeffdingTree
from skmultiflow.drift_detection import DDM
from skmultiflow.data import DataStream
# inicializa classificador 1
clf1 = HoeffdingTree()
# treina classificador 1
clf1.partial_fit(train_features_norm, train_label)
# inicializa classificador 2
clf2 = HoeffdingTree()
# inicializa detector de drift
drift = DDM()
# cria stream com dados de teste
s = DataStream(test_features_norm, test_label)
# prepara stream para uso
s.prepare_for_use()
```

Código 19. Inicializando dois classificadores (árvore de decisão *HoeffdingTree*), um detector de *drift* e a *stream* para classificar um conjunto de testes criado a partir do *dataset Brazilian Malware*.

O Código 21 apresenta um exemplo de como criar uma rede neural *Multi-Layer Perceptron* com duas camadas escondidas. Primeiramente, é necessário transformar o vetor de rótulos no formato *one-hot encoding*, similar ao utilizado pelo *Word2Vec*. Então, inicializá-se o modelo do tipo `Sequential` e, logo em seguida, podemos adicionar camadas a este modelo. A primeira camada nesse exemplo contém 200 neurônios e é necessário especificar o tamanho da entrada (o número de características), assim como a função de ativação (estamos utilizando a função *relu*). Em seguida, adicionamos uma segunda camada, com 100 neurônios e a mesma função de ativação. Finalmente, adicionamos a camada de saída, composta por dois neurônios (um para cada classe do problema) e a função *softmax*, para transformar seus valores em probabilidades. O modelo então é compilado, especificando a função de minimização (no caso *binary_crossentropy*), otimizador (*rmsprop*) e métricas a serem observadas durante o treino, neste caso, a acurácia. Depois de compilado, o modelo pode ser treinado usando os dados de treino e os rótulos de treino transformados em *one-hot encoding*. No exemplo, especificamos um conjunto de validação de 33%, isto é, 33% dos dados de treino serão utilizados para validar o modelo enquanto ele é treinado em 10 épocas e em *batches* de 128 exemplos (o modelo é atualizado durante o treino a cada 128 exemplares). O método `predict_classes` prediz as classes do conjunto de testes. A documentação da biblioteca, assim como as demais, apresenta mais exemplos de redes, bem como todos os seus parâmetros e detalhes.

2.6. Avaliação de Modelos

Tão importante quanto criar uma boa representação para o problema e criar um bom modelo, saber avaliar se o mesmo resolve o problema em questão é fundamental, sobretudo na área de segurança. É comum encontrar na literatura trabalhos que tentam apenas obter

```
from skmultiflow.core import clone
# cria vetor final de predicao
test_pred = []
# contador de dados
count = 0
# armazena pontos de drift
drifts = []
# itera sobre stream
while s.has_more_samples():
    # obtem caracteristicas e rotulo do proximo exemplar
    sample_features, sample_label = s.next_sample(1)
    # incrementa contador
    count += 1
    # prediz exemplar
    sample_pred = clf1.predict(sample_features)
    # adiciona predicao ao vetor de predicao
    for l in sample_pred:
        test_pred.append(l)
    # adiciona resultado da predicao ao detector
    for e in sample_label == sample_pred:
        drift.add_element(e)
    # detecta se houve warning ou drift
    if drift.detected_warning_zone():
        # atualiza classificador 2
        clf2.partial_fit(sample_features, sample_label)
    if drift.detected_change():
        # salva ponto de drift
        drifts.append(count)
        # muda classificador 1 pelo classificador 2
        clf1 = clone(clf2)
        # reseta classificador 2
        clf2 = HoeffdingTree()
        # reseta detector de drift
        drift.reset()
    # atualiza modelo com novo exemplar
    clf1.partial_fit(sample_features, sample_label)
# imprime pontos de drift
print("Drifts in {}.".format((drifts)))
```

```
Drifts in [19936, 20938, 21126, 21183, 24425, 24493,
  ↪ 24536, 24729].
```

Código 20. Utilizando um detector de *drift* para classificar um conjunto de testes criado a partir do *dataset Brazilian Malware*

uma acurácia próxima de 100% em problemas de classificação de *malware*, isto é, eles avaliam somente se todos os exemplares, independente de sua classe e distribuição de dados, foram corretamente classificados, o que no fundo pode não ser uma forma adequada de avaliar o problema [Ceschin et al. 2018]. Por exemplo, considere que um modelo é avaliado utilizando dez exemplares: oito deles são benignos e dois, malignos. Este modelo tem acurácia de 80%. Seria 80% uma acurácia satisfatória? Supondo que o modelo acerte apenas os oito exemplares benignos, ele não é capaz de identificar nenhum exemplo maligno e mesmo assim a medida de acurácia atinge 80%, passando a falsa impressão de que o modelo funciona bem. É necessário lembrar que, em problemas binários em

```
import keras
from keras.models import Sequential
from keras.layers import Dense
# converte rotulos para a codificacao one-hot encoding
train_label_onehot =
  ↪ keras.utils.to_categorical(train_label,
  ↪ num_classes=2)
test_label_onehot =
  ↪ keras.utils.to_categorical(test_label,
  ↪ num_classes=2)
# inicializa rede sequencial
model = Sequential()
# adiciona camada escondida com 200 neuronios
model.add(Dense(200, activation='relu',
  ↪ input_dim=train_features_norm.shape[1]))
# adiciona camada escondida com 100 neuronios
model.add(Dense(100, activation='relu'))
# camada de saida
model.add(Dense(2, activation='softmax'))
# compila modelo
model.compile(loss='binary_crossentropy',
  ↪ optimizer='rmsprop', metrics=['accuracy'])
# treina modelo com dados de treino
model.fit(train_features_norm, train_label_onehot,
  ↪ validation_split=0.33, epochs=10, batch_size=128,
  ↪ verbose=0)
# prediz classes
test_pred = model.predict_classes(test_features_norm)
# imprime forma da predicao e dos rotulos reais de teste
print(test_pred.shape, test_label.shape)
```

```
(25090,) (25090,)
```

Código 21. Utilizando o classificador *Multi-Layer Perceptron* para classificar um conjunto de testes criado a partir do *dataset Brazilian Malware*.

geral, um modelo de aprendizado de máquina deve ser robusto o suficiente para identificar padrões de forma genérica, independente da classe utilizada, já que não queremos um modelo que classifique tudo como sendo de apenas uma única classe. Nesta Seção apresentaremos as métricas mais comuns utilizadas em problemas de segurança, bem como métodos de validar as soluções.

2.6.1. Métricas

Para poder avaliar corretamente um modelo, deve-se escolher corretamente uma métrica, já que cada uma pode apresentar uma perspectiva do problema (e até mesmo dar falsas impressões sobre o resultado geral do problema, como no exemplo mencionado anteriormente). Neste curso apresentaremos as seguintes métricas: acurácia, matriz de confusão, *recall*, *precision* e *f1score*.

2.6.1.1. Acurácia

A acurácia é uma medida que mede a porcentagem de exemplares que o classificador conseguiu acertar de determinado conjunto (normalmente do conjunto de testes). Basicamente, corresponde ao número de exemplares corretamente classificados dividido pelo número de exemplares apresentados ao classificador. A biblioteca *Scikit-Learn* implementa a acurácia através do método `accuracy_score`[31], bastando passar como parâmetro as predições feitas pelo classificador e os rótulos do conjunto de teste utilizado (classes reais do conjunto), como mostrado no Código 22. Esta métrica não é recomendada para problemas cujo conjunto de dados é desbalanceado, uma vez que ela pode ter um alto índice, mesmo quando o classificador favorece a classe majoritária e erra todos os exemplares de classes minoritárias [Gron 2017].

```
1 from sklearn.metrics import accuracy_score
2 print(accuracy_score(test_label, test_pred))
```

```
1 0.8836986847349542
```

Código 22. Acurácia do *Random Forest* para classificar um conjunto de teste criados a partir do *dataset Brazilian Malware*.

2.6.1.2. Matriz de Confusão

A matriz de confusão é uma forma de melhor visualizar os resultados gerados por um classificador. Seja C uma matriz de confusão, $C_{i,j}$ é igual ao número de observações em que o classificador considerou um exemplar da classe i como sendo da classe j. Em classificação binária podemos extrair informações da matriz de confusão: os verdadeiro negativos (*True Negatives – TN*), que são $C_{0,0}$, os falsos negativos (*False Negatives – FN*), que são

[31] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html`

$C_{1,0}$, verdadeiros positivos (*True Positives – TP*), que são $C_{1,1}$ e falsos positivos (*False Positives – FP*), que são $C_{0,1}$ [Pedregosa et al. 2011]. O Código 23 apresenta a matriz de confusão para o nosso problema de exemplo, bastando utilizar o método `confusion_matrix`[32] da *Scikit-Learn*, passando como parâmetro as predições e as classes reais dos dados de teste utilizados. É possível observar que o número de verdadeiros negativos é de 6987 ($C_{0,0}$), o de falsos negativos é de 2672 ($C_{1,0}$), o de verdadeiros positivos é de 15185 ($C_{1,1}$) e o de falsos positivos é de apenas 46 ($C_{0,1}$). A partir dessas informações, duas novas medidas podem ser extraídas (*recall* e *precision*), gerando uma terceira métrica que utiliza ambas (*f1score*).

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(test_label, test_pred))
```

```
[[ 6987    46]
 [ 2872 15185]]
```

Código 23. Matriz de confusão do *Random Forest* para classificar um conjunto de teste criados a partir do *dataset Brazilian Malware*.

2.6.1.3. *Recall*

O *recall*, também chamado de *sensitivity* ou *true positive rate* (*TPR*), é a proporção de instâncias positivas que são corretamente classificadas pelo classificador, isto é, a habilidade do classificador encontrar todos os exemplares positivos [Gron 2017, Pedregosa et al. 2011]. O mesmo utiliza o número de verdadeiros positivos (*True Positives – TP*) e falsos negativos (*False Negatives – FN*) e é dado pela Equação 13. O *Scikit-Learn* implementa essa métrica através do método `recall_score`[33], passando como referência o vetor de predição e o vetor de classes reais, como mostrado no Código 24. Em nosso problema exemplo, obtemos cerca de 84% de *recall*, indicando que o classificador confunde alguns *malware* como sendo *goodware* (cerca de 16%).

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

2.6.1.4. *Precision*

A *precision* é uma métrica capaz de medir a acurácia das predições positivas do classificador, isto é, ela mede a habilidade do classificador não rotular como positivo um exemplar que é negativo [Gron 2017, Pedregosa et al. 2011]. Essa métrica utiliza o número de verdadeiros positivos (*True Positives – TP*) e o número de falsos positivos (*False Positives*

[32] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`

[33] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html`

```
from sklearn.metrics import recall_score
print(recall_score(test_label, test_pred))
```

```
0.8409481087666832
```

Código 24. *Recall* do *Random Forest* para classificar um conjunto de teste criados a partir do *dataset Brazilian Malware*.

– *FP*) e é dada pela Equação 14. O *Scikit-Learn* também implementa essa métrica através do método `precision_score`[34], passando como parâmetro o vetor de predição e o vetor de classes reais, como mostrado no Código 25. No exemplo, obtemos cerca de 99% de *precision*, indicando que poucos *goodware* estão sendo confundidos como sendo *malware* (menos de 1% são confundidos).

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

```
from sklearn.metrics import precision_score
print(precision_score(test_label, test_pred))
```

```
0.9969798437397414
```

Código 25. *Precision* do *Random Forest* para classificar um conjunto de teste criados a partir do *dataset Brazilian Malware*.

2.6.1.5. *F1Score*

O *F1Score* utiliza tanto o *recall* como a *precision* para gerar uma única métrica, que nada mais é do que a média harmônica entre essas duas métricas, dado pela Equação 15 [Gron 2017, Pedregosa et al. 2011]. O *Scikit-Learn* implementa essa métrica através do método `f1_score`[35], passando como parâmetro o vetor de predição e o vetor de classes reais, como mostrado no Código 26. Em nosso problema exemplo, obtemos cerca de 91% de *f1score*.

$$F1Score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{15}$$

Embora sera interessante, é sempre bom reportar o *recall* e a *precision* individualmente, uma vez que em determinados problemas, um pode ser mais importante que o

[34] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html`

[35] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`


```
from sklearn.metrics import f1_score
print(f1_score(test_label, test_pred))
```

```
0.9123407834655132
```

Código 26. *F1Score* do *Random Forest* para classificar um conjunto de teste criados a partir do *dataset Brazilian Malware*.

outro e o *F1Score* pode não captar essa necessidade, já que é apenas uma média entre os dois [Gron 2017]. Para um detector de *malware*, por exemplo, muitas vezes pode ser preferível que um *malware* não seja detectado do que um software benigno seja bloqueado (alta *precision*), já que afetaria diretamente na experiência de uso de um usuário. Imagine que um usuário esteja usando seu computador e, ao abrir o *Microsoft Word*, o classificador acredita que ele seja um *malware* e bloqueia a execução do programa. Por mais que o classificador detecte grande maioria dos *malware*, ele torna o computador inutilizável pelo usuário, já que bloquearia grande partes das aplicações benignas do usuário. Em contrapartida, em alguns sistemas mais sensíveis, pode-se querer o contrário (alto *recall*), bloqueando todas as ações malignas, mesmo que algumas benignas sejam "sacrificadas". Essas configurações podem ser feitas utilizando probabilidades de cada classe na saída do classificador, bastando definir um limiar (*threshold*) para que o exemplar seja de uma classe ou de outra, como mostrado no Código 27. Em relação aos resultados anteriores, nota-se que a *precision* aumentou um pouco e o *recall* caiu significativamente, isto é, o número de *malware* classificado como *goodware* aumentou muito em troca de um pequeno número de *goodware* deixando de ser confundido com *malware*. É importante ressaltar que a medida que você aumenta a *precision*, o *recall* diminui e vice versa. Este problema é conhecido como *precision/recall tradeoff* [Gron 2017]. A Figura 2.15 apresenta exatamente esse *tradeoff* entre *recall* e *precision* para o problema exemplo proposto, variando o *threshold* de 0 a 100%.

```
# valor do threshold
threshold = 0.7
# obtem probabilidade de cada exemplar
test_proba = clf.predict_proba(test_features_norm)
# cria um vetor com as classes, aplicando threshold
test_pred = (test_proba[:,1] > threshold).astype('int')
# imprime recall e precision
print("Rec:", recall_score(test_label, test_pred))
print("Pre:", precision_score(test_label, test_pred))
```

```
Rec: 0.7345073932546935
Pre: 0.9975930801053028
```

Código 27. *Recall* e *precision* utilizando um *threshold* de 70% para classificar um conjunto de teste criados a partir do *dataset Brazilian Malware*.

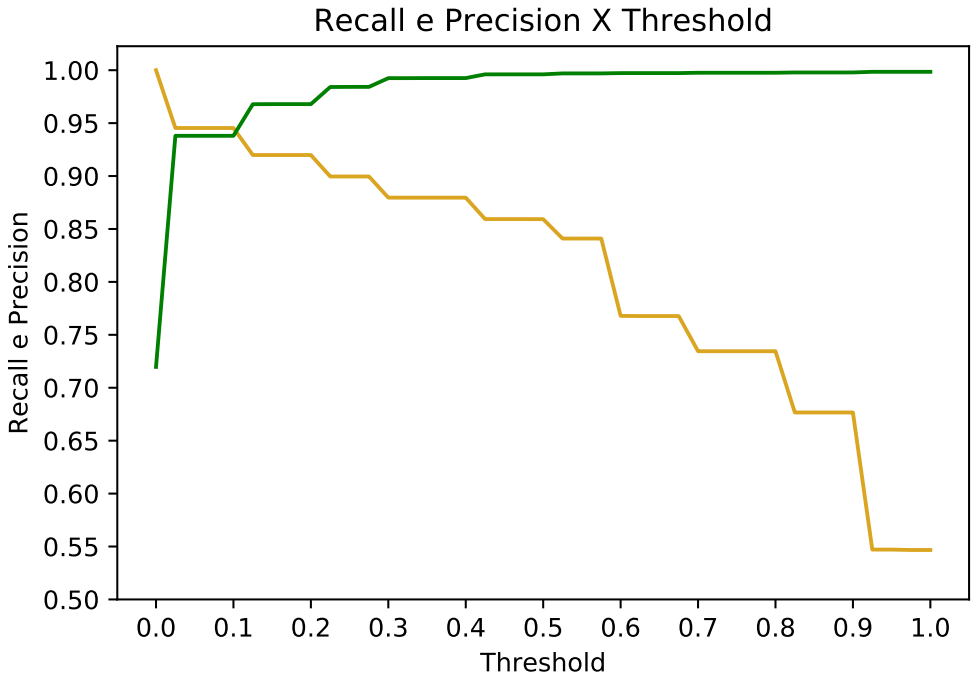


Figura 2.15. *Tradeoff* do *recall* e *precision* para classificar um conjunto de teste criados a partir do *dataset Brazilian Malware*.

2.6.2. Validação

A validação de um modelo é importante pois define o quão genérico o mesmo é, evitando um fenômeno chamado de *overfitting*: quando um classificador "decora" os dados utilizados no treino, obtendo altas taxas de acurácia, porém sem generalizar o problema, isto é, ao ser utilizado de fato no mundo real, o classificador passa a atuar de forma muito diferente do esperado, errando muito mais do que o observado [Gron 2017]. Para problemas em que os dados estão disponíveis em *batches*, é muito comum utilizar o um método chamado de *K-Fold Cross Validation*, que consiste basicamente em dividir aleatoriamente os dados em K subconjuntos distintos chamados de *folds* e treinar e avaliar o modelo utilizado K vezes, utilizando um *fold* diferente para a avaliação (teste) cada vez e os outros *K-1* para treino. O Código 28 apresenta um exemplo de *K-Fold Cross Validation*, utilizando o método `cross_val_score`[36] da biblioteca *Scikit-Learn*, em que o primeiro parâmetro é o classificador a ser utilizado, o segundo é o conjunto de características, o terceiro é o conjunto de rótulos dessas características, *cv* é o número de *folds* criados e *scoring* é a métrica a ser utilizada. Pelos resultados, observa-se que as acurácias em nosso problema variaram de cerca de 61%, no primeiro fold, até o máximo de quase 99%, atingindo um valor médio de 94% de acurácia.

Para dados em *stream*, já mencionamos a forma correta de se validar um modelo, já que o mesmo simula um classificador que recebe dados em "tempo real", se adaptando à mudanças conforme o tempo passa. Geralmente não é possível utilizar o *K-Fold Cross Validation* em problemas de *stream*, pois o mesmo acaba misturando de forma aleatória

[36] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html`


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
# inicializa classificador
clf = RandomForestClassifier(n_estimators=10)
# obtem resultados
results = cross_val_score(clf, train_features_norm,
  ↪ train_label, cv=10, scoring="accuracy")
# imprime vetor de acuracia por fold
print(results)
# imprime media dos resultados
print(np.mean(results))
```

```
[0.61195219 0.92151394 0.99561753 0.99282583 0.99521722
  ↪ 0.99163013 0.9920287  0.96811479 0.98883573
  ↪ 0.97886762]
0.9436603674644856
```

Código 28. Resultados do *K-Fold Cross Validation* utilizando o classificador *Random Forest* para classificar um conjunto de dados criados a partir do *dataset Brazilian Malware*.

os exemplares de épocas (e conceitos) diferentes, o que não torna o resultado real: é mais fácil para um classificador "prever o futuro" se ele já viu um exemplo similar ao da época em que vai ser avaliado. Por exemplo, supondo que os *malware* do tipo *Ransomware* ficaram comuns em 2017, avalia-se se um modelo é capaz de detectar esse tipo de ameaça utilizando dados coletados de 2015 a 2017. Ao utilizar o *K-Fold Cross Validation*, é possível que os exemplares de 2017, contendo *RansomWare*, sejam inseridos em algum momento durante o treino, fazendo com que o modelo funcione bem e obtenha uma ótima métrica. Entretanto, na realidade, o classificador não teria nenhum exemplar semelhante até que os mesmos fossem de fato desenvolvidos e conhecidos, tornando o problema seja muito mais difícil.

2.7. Conclusão

O aprendizado de máquina pode ser um poderoso aliado a segurança computacional se utilizado corretamente. Entretanto, ainda é comum encontrar soluções cujos resultados não condizem com a realidade. Por conta disso, é necessário ter certeza de que a solução proposta realmente funciona no mundo real, avaliando-a da forma correta, isto é, levando em consideração o fato de que os dados podem não ter distribuição estacionária e estar em constante mudança.

Neste capítulo, apresentamos diversos conceitos de aprendizado de máquina fundamentais para a área de segurança. Começamos com a coleta dos dados, na Seção 2.2, mostrando como realizá-la e rotular corretamente, os principais problemas relacionados a isso e dois exemplos de *datasets*. Na sequência, apresentamos técnicas de extração de

atributos estáticas de Windows e Android e recomendamos ferramentas para a análise dinâmica dos mesmos, na Seção 2.3. Então, na Seção 2.4, mostramos dois algoritmos de extração de características (TF-IDF e *Word2Vec*) e dois métodos de normalização das mesmas (*min-max* e *standardization*). Em seguida, os principais classificadores, detectores e agrupadores foram apresentados, com seus respectivos códigos e bibliotecas, destacando como se utilizar uma *stream* e detectores de *drift*, na Seção 2.5. Para finalizar, mostramos as principais métricas para avaliar corretamente os modelos utilizados, bem como as formas de avaliá-los, incluindo até mesmo uma técnica para poder definir um limiar para os resultados do classificador, na Seção 2.6.1.

Deste modo, esperamos que este trabalho tenha um impacto positivo na comunidade acadêmica e na indústria, fomentando melhorias nas soluções que envolvem ambos os temas (aprendizado de máquina e segurança) e servindo de base para estudos introdutórios ou mesmo confecção de cursos mais avançados.

Referências

[Abadi et al. 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from `tensorflow.org`.

[Albert Bifet 2018] Albert Bifet, Ricard Gavalda, G. H. B. P. (2018). *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press. `https://moa.cms.waikato.ac.nz/book/`.

[Almeida et al. 2015] Almeida, P., Oliveira, L., Britto, A., and Sabourin, R. (2015). Dealing with concept drifts using dynamic ensembles of classifiers. Tesis presented as partial requirement for the degree of Doctor. Graduate Program in Informatics, Sector of Exact Sciences, Universidade Federal do Paraná.

[Arp et al. 2014] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., and Rieck, K. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*.

[Baena-Garćıa et al. 2006] Baena-Garćıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., and Morales-Bueno, R. (2006). Early drift detection method.

[Bifet and Gavaldà 2007] Bifet, A. and Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. volume 7.

[Bishop 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Botacin et al. 2018] Botacin, M., de Geus, P. L., and Grégio, A. (2018). The other guys: automated analysis of marginalized malware. *Journal of Computer Virology and Hacking Techniques*, 14(1):87–98.

[Breiman 2001] Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

[Ceschin et al. 2018] Ceschin, F., Pinage, F., Castilho, M., Menotti, D., Oliveira, L. S., and Gregio, A. (2018). The need for speed: An analysis of brazilian malware classifers. *IEEE Security Privacy*, 16(6):31–41.

[Chollet et al. 2015] Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

[Cortes and Vapnik 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Dwork and Roth 2014] Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407.

[Ester et al. 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*.

[Filho et al. 2011] Filho, D. S. F., Afonso, V. M., Martins, V. F., Grégio, A. R. A., de Geus, P. L., Jino, M., and dos Santos, R. D. C. (2011). Técnicas para análise dinâmica de malware. *Minicurso do SBSEG*. `https://sbseg2011.redes.unb.br/resources/downloads/minicursos/91936.pdf`.

[Fradkin and Muchnik 2006] Fradkin, D. and Muchnik, I. (2006). Support vector machines for classification. *"Discrete Methods in Epidemiology", DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 70:13–20.

[Gama et al. 2004] Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. In Bazzan, A. L. C. and Labidi, S., editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Gama et al. 2014] Gama, J. a., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37.

[Gandotra et al. 2014] Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 5(2):56–64.

[Gron 2017] Gron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition.

[Haykin 2009] Haykin, S. (2009). *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall.

[Hoffmann et al. 2016] Hoffmann, J., Rytilahti, T., Maiorca, D., Winandy, M., Giacinto, G., and Holz, T. (2016). Evaluating analysis tools for android apps: Status quo and robustness against obfuscation. pages 139–141.

[Jordan 2017] Jordan, M. I. (2017). The kernel trick, advanced topics in learning & decision making. `https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf`, accessed in July 2017.

[Manning et al. 2008a] Manning, C. D., Raghavan, P., and Schütze, H. (2008a). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Manning et al. 2008b] Manning, C. D., Raghavan, P., and Schütze, H. (2008b). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

[Mellish 2017] Mellish, C. (2017). Machine learning, lecture notes. `http://www.inf.ufpr.br/lesoliveira/aprendizado/machine_learning.pdf`, accessed in July 2017.

[Michie et al. 1994] Michie, D., Spiegelhalter, D. J., Taylor, C. C., and Campbell, J., editors (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA.

[Mikolov et al. 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mikolov et al. 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

[Milgram et al. 2006] Milgram, J., Cheriet, M., and Sabourin, R. (2006). "One Against One" or "One Against All": Which One is Better for Handwriting Recognition with SVMs? In Lorette, G., editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France). Université de Rennes 1, Suvisoft. http://www.suvisoft.com.

[Mitchell 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Montiel et al. 2018] Montiel, J., Read, J., Bifet, A., and Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5.

[Oktavianto and Muhardianto 2013] Oktavianto, D. and Muhardianto, I. (2013). *Cuckoo Malware Analysis*. Packt Publishing.

[Pedregosa et al. 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Řehůřek and Sojka 2010] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

[Rogers and Girolami 2011] Rogers, S. and Girolami, M. A. (2011). *A First Course in Machine Learning*. Chapman and Hall / CRC machine learning and pattern recognition series. CRC Press.

[Rong 2014] Rong, X. (2014). word2vec parameter learning explained. *CoRR*, abs/1411.2738.

[Saxe and Sanders 2018] Saxe, J. and Sanders, H. (2018). *Malware Data Science: Attack Detection and Attribution*. No Starch Press, San Francisco, CA, USA.

[Schubert et al. 2017] Schubert, E., Sander, J., Ester, M., Kriegel, H. P., and Xu, X. (2017). Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3):19:1–19:21.

[Sebastián et al. 2016] Sebastián, M., Rivera, R., Kotzias, P., and Caballero, J. (2016). Avclass: A tool for massive malware labeling. In Monrose, F., Dacier, M., Blanc, G., and Garcia-Alfaro, J., editors, *Research in Attacks, Intrusions, and Defenses*, pages 230–253, Cham. Springer International Publishing.

[Seide and Agarwal 2016] Seide, F. and Agarwal, A. (2016). Cntk: Microsoft's open-source deep-learning toolkit. pages 2135–2135.

[Shulman 2016] Shulman, B. (2016). A tour of sentiment analysis techniques: Getting a baseline for sunny side up. https://gab41.lab41.org/a-tour-of-sentiment-analysis-techniques-getting-a-baseline-for-sunny-

[Symantec 2019] Symantec (2019). 2019 internet security threat report. https://www.symantec.com/security-center/threat-report.

[Tam et al. 2017] Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., and Cavallaro, L. (2017). The evolution of android malware and android analysis techniques. *ACM Comput. Surv.*, 49(4):76:1–76:41.

[Theano Development Team 2016] Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

[Wang et al. 2011] Wang, S., Schlobach, S., and Klein, M. (2011). Concept drift and how to identify it. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(3):247 – 265. Semantic Web Dynamics Semantic Web Challenge, 2010.

[Yonts 2010] Yonts, J. (2010). *Building a Malware Zoo*. The SANS Institute.

Capítulo 3

Análise de mecanismos para consenso distribuído aplicados a Blockchain

Charles C. Miers (UDESC), Guilherme P. Koslovski (UDESC),
Maurício A. Pillon (UDESC), Marcos A. Simplício Jr. (USP),
Tereza C. M. B. Carvalho (USP), Bruno B. Rodrigues (UZH),
João H. F. Battisti (UDESC)

Abstract

The Blockchain concept has recently emerged as an alternative approach for e-commerce payment, based on decentralized systems that do not rely on trusted institutions. Actually, since Blockchain was first proposed for use in cryptocurrencies, several solutions have appeared that employ this technology in a variety of domains, such as creating distributed registries of smart contracts. Whichever the target application domain, though, solutions implementing Blockchain use a set of well-known technologies, such as encryption, Merkle Trees, P2P networks, and consensus mechanisms. The latter are of particular research interest today, since most consensus mechanisms used in the early cryptocurrencies (e.g., Bitcoin) involve considerable computational power to ensure system consistency. Aiming to evaluate the state of the art on the area, this manuscript reviews Blockchain concepts and classifications, focusing specifically on the underlying consensus mechanisms and security aspects of the resulting solutions. We also describe a brief experiment using Ethereum and MultiChain, aiming to analyze security aspects of the Practical Byzantine Fault Tolerance (PBFT) and Proof-of-Work (PoW) consensus approaches.

Resumo

O conceito de Blockchain tem emergido como um meio alternativo para pagamentos em cenários de comércio eletrônico, criando um sistema descentralizado no qual não existe dependência de instituições confiáveis. Além disso, desde que foi originalmente proposto para uso em criptomoedas, diversas soluções surgiram que visam empregar a tecnologia de Blockchain em uma variedade de domínios de aplicação, como a criação de um cartório distribuído para armazenar contratos inteligentes. A despeito do domínio de aplicação específico, porém, soluções baseadas em Blockchain utilizam um conjunto de

diversas tecnologias bem conhecidas, como esquemas criptográficos, árvores de Merkle, redes P2P e mecanismos de consenso. Estes últimos são de particular interesse de pesquisa atualmente, em especial porque a maioria dos mecanismos de consenso usados nas primeiras criptomoedas (e.g., Bitcoin) exigem um poder computacional considerável para garantir a consistência do sistema. Com o objetivo de avaliar o estado da arte das pesquisas nesta área, o presente manuscrito apresenta os principais conceitos e classificações de Blockchain, dando ênfase nos mecanismo de consenso subjacentes e nos aspectos de segurança das soluções resultantes. Também é apresentado um breve experimento usando as soluções Ethereum e MultiChain para ilustrar os aspectos de segurança relativos aos consensos do tipo Tolerância Prática a Falhas Bizantinas (Practical Byzantine Fault Tolerance – PBFT) e Prova-de-Trabalho (Proof-of-Work – PoW).

3.1. Introdução

O grande volume de transações eletrônicas realizadas atualmente realça a sua importância nas relações comerciais modernas. Ao mesmo tempo, essa relevância lança desafios diversos, tais como a necessidade de desenvolvimento de tecnologias que forneçam maior eficiência e segurança nas transações realizadas. Neste sentido, o uso de soluções que sigam um paradigma centralizado (e.g., banco de dados tradicionais) muitas vezes tem sua eficiência e segurança questionadas, seja por gerar pontos centrais de falha ou por questões de escalabilidade e confiabilidade. Em parte devido a essas preocupações, surgiu recentemente um movimento cujo objetivo é elaborar novos conceitos e soluções baseados em abordagens não-centralizadas. Cabe notar que os bancos e instituições financeiras tradicionais, em sua grande maioria, são exemplos de centralização quando refere-se ao modo como as transações financeiras são realizadas: estas concentram a confiança durante as transações e centraliza o banco de dados com informações financeiras, criando assim uma relação de dependência por parte seus usuários [1, 2].

Um banco de dados tradicional geralmente usa uma arquitetura cliente-servidor, na qual as entradas feitas pelos clientes são armazenadas em um servidor centralizado e, eventualmente, replicadas. Nesse caso, o controle do banco de dados é mantido por uma autoridade central no lado do servidor, que regula o acesso e decide quais entidades têm permissão de leitura ou gravação. Em contraste, um Blockchain usa um modelo totalmente descentralizado, no qual cada participante (não necessariamente confiável) mantém, calcula e atualiza novas entradas que são replicadas em todos os nós do sistema. Os registros das transações formam então uma espécie de *livro razão distribuído* (usando o termo comumente empregado na área de contabilidade), que pode ser acessado por qualquer usuário para verificar a validade das transações realizadas, segundo regras definidas pela aplicação. É exatamente essa característica do Blockchain que é explorada no Bitcoin, a primeira criptomoeda criada com base nessa tecnologia e a responsável por trazer maior notoriedade ao conceito de Blockchain. Tal notoriedade deve-se ao fato do Bitcoin criar um cenário financeiro com grande autonomia, no qual transações comerciais eletrônicas não dependem do aval de instituições financeiras governamentais ou privadas reconhecidas, e ainda assim podem ser feitas com um bom grau de segurança [3]. Em particular, o Blockchain no Bitcoin previne tentativas de *double-spending* (i.e., "gasto-duplo"), em que um usuário desonesto tenta repassar um mesmo conjunto de moedas para mais de um usuário.

Embora o Blockchain tenha sido criado para permitir transações financeiras, e seja comumente associado a esse domínio, outras áreas de aplicação e uso de Blockchain estão surgindo com o crescimento da atenção do público em torno da tecnologia [4]. De fato, as tecnologias que possibilitam a implementação Blockchain têm sido objeto de um número crescente de pesquisas científicas, gerando interesse significativo por diversos setores da indústria, governamental e pesquisadores devido às suas características de transparência, confiança e segurança [5]. Como resultado, enquanto a primeira implementação popular do Blockchain (o Bitcoin) foi introduzida em 2008 [6], desde então vários sistemas Blockchain, como Ethereum [7], Hyperledger [8] e Multichain [9] emergiram com propostas fora do setor financeiro [3].

Embora os detalhes da implementação do Blockchain possam variar dependendo da aplicação alvo, a tecnologia em si se baseia em dois elementos subjacentes principais:

- Uma estrutura de dados com verificação de integridade: como o próprio nome indica, o Blockchain utiliza uma estrutura de dados baseada em encadeamento de blocos, em que cada bloco (um conjunto de registros quaisquer, acompanhados de metadados) carrega o valor de *hash* do bloco anterior. Ao assinar digitalmente os registros armazenados nos blocos individuais, o Blockchain permite a detecção de alterações locais e, indiretamente, também permite verificar a integridade dos blocos anteriores (cujos *hashes* seriam alterados no caso de modificações indevidas).
- Um mecanismo de consenso distribuído: o Blockchain utiliza técnicas para garantir a consistência do livro-razão, i.e., que todos os usuários eventualmente terão a mesma visão sobre a ordem em que os blocos foram inseridos na cadeia. Assim, garante-se que a característica distribuída do banco de dados não leve a decisões inconsistentes por parte dos usuários. Em particular, esse mecanismo é importante para evitar que uma mesma moeda seja gasta múltiplas vezes: contanto que todos os usuários concordem sobre qual foi a primeira transação de repasse dessa moeda, apenas essa primeira transação será considerada válida. Ao realizar esse procedimento, o Blockchain pode ser visto como uma Autoridade de Carimbo de Tempo (*Timestamp Authority* – TA) distribuída, que define a relação temporal entre bloco inserido no sistema. É importante notar, entretanto, que essa relação temporal é: (1) apenas relativa, i.e., não se busca determinar os instantes de tempo exatos em que uma transação ocorreu; e (2) não necessariamente corresponde aos instantes de tempo reais, de modo que uma transação que ocorra no instante de tempo t pode aparecer no Blockchain depois de uma transação que ocorra posteriormente, em um instante $t + \varepsilon$. Portanto, o Blockchain cria uma espécie de "linha de tempo alternativa", que não necessariamente corresponde à ordem temporal real das transações.

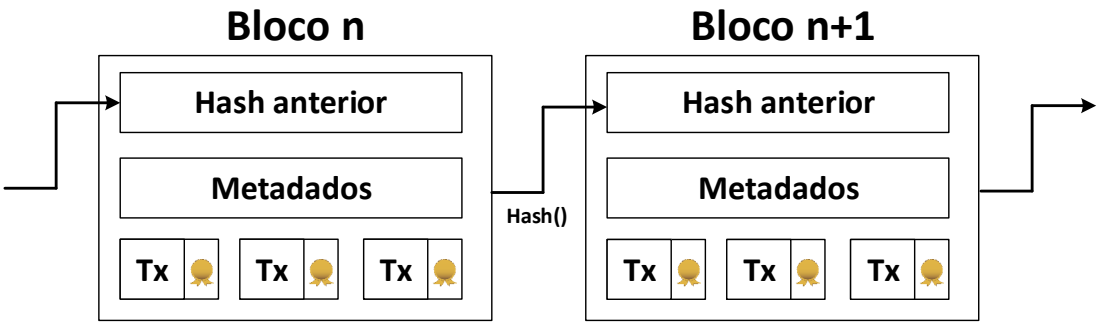
Cada um desses elementos básicos é discutido mais profundamente nas próximas seções. Antes disso, entretanto, é importante ressaltar que o Blockchain *per se* não garante a validade dos dados subjacentes nele introduzidos, mas apenas que o conteúdo e a ordem dos blocos contendo esses dados não podem ser alterados de forma imperceptível. Assim, a verificação da validade dos dados fica à cargo da camada de aplicação, que deve definir as regras para que os dados inseridos nos blocos sejam considerados "válidos". No caso do Bitcoin, a regra básica é que (1) as moedas sendo transferidas ainda estejam na

posse do usuário origem da transferência (i.e., não exista uma transação anterior em que a moeda tenha sido gasta), e (2) a transferência seja digitalmente assinada pela origem, garantindo sua anuência com relação à transferência. Para outras aplicações, as regras em questão devem ser definidas conforme a necessidade. Por exemplo, em aplicações cujo objetivo é permitir a troca de ativos genéricos entre usuários (i.e., não apenas ativos financeiros virtuais), dispensando entidades confiáveis e processos cartoriais, é comum o uso de regras similares às do Bitcoin para verificação da posse do ativo e da assinatura do seu dono na transação. Entretanto, no caso de ativos do mundo real, normalmente é também necessário "virtualizá-lo", ou seja, identificá-lo de forma unívoca e verificável tanto no mundo digital e real. Isso normalmente passa pela criação de um *identificador virtual* que garanta a própria existência do ativo e sua descrição; alguns exemplos de ativos cuja virtualização é possível incluem imóveis, que podem ser identificados pelo seu número de matrícula, e veículos, que podem ser identificados pelo seu número de chassis. Cabe notar que, embora esse processo de identificação possa envolver um entidade confiável (e.g., um cartório), as transações posteriores de transferência desse ativo podem ser feitas sem a interveniência dessa entidade, bastando incluir o identificador do ativo entre os dados da transação. Transações fraudulentas ainda podem ocorrer, porém o histórico de eventos permite identificar eventuais transgressões.

3.1.1. A estrutura de dados do Blockchain

A estrutura de dados subjacente a esquemas de Blockchain consiste em uma sequência de blocos, como ilustrado na Figura 3.1. Nesta sequência, fica armazenada uma lista completa de registros, criando um conjunto de dados que pode ser consultado por qualquer usuário do sistema [6]. A semântica de cada registro é definida pela aplicação em si, dependendo das necessidades do cenário alvo, cabendo ao Blockchain apenas armazenar esses registros com garantias de integridade dos blocos e de sua ordem relativa no sistema. No entanto, para permitir uma discussão mais focada e sem perda de generalidade, ao longo deste documento assume-se que os registros referem-se a transações de ativos entre diferentes partes. A escolha desse cenário deve-se ao fato de este ser um cenário típico no qual é necessária a ordenação de registros (no caso, para determinar o proprietário atual), que é a finalidade básica de um Blockchain.

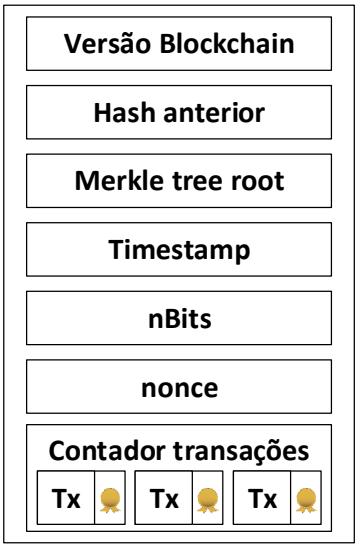
Figura 3.1: Blockchain: exemplo de sequência de blocos.



Os principais elementos de cada bloco em um Blockchain, que também são mostrados na Figura 3.1, são: (1) o valor do *hash* do conteúdo completo do bloco anterior, criando uma ordenação entre eles e permitindo a verificação da integridade de cada bloco e também de sua ordem relativa; e os dados da aplicação em si, representados na Figura 3.1 como uma lista de transações Tx, e que comumente são assinados digitalmente pelo usuário responsável por sua geração, ou mesmo por todas as partes envolvidas na transação. Como metadados adicionais, o bloco completo costuma incluir alguma forma de identificação da(s) entidade(s) que fizeram a seleção e verificação dos dados armazenados no bloco, entidades estas conhecidas como *validadores* ou *mineradores*[1]. Essa identificação costuma se dar na forma da chave pública do(s) minerador(es), que pode ser acompanhada ou não da assinatura digital do bloco completo; no Bitcoin, por exemplo, a primeira transação de qualquer bloco, chamada de *transação base* (do inglês *coinbase transaction*) consiste basicamente na chave pública do minerador, a qual não acompanha sua assinatura digital. Outros metadados vão depender da aplicação específica do Blockchain. Por exemplo, algumas soluções incluem um carimbo de tempo (*timestamp*) com a data e a hora da criação do bloco; entretanto, essa informação só deve ser considerada para fins de referência, pois o baixo grau de sincronismo normalmente apresentado por uma rede Blockchain impediria o registro de um instante exato e confiável de tempo.

A Figura 3.2 exemplifica a composição de um bloco genérico no Bitcoin. Em suma, os seguintes campos estão presentes nos blocos em adição às transações escolhidas pelos mineradores [10]:

Figura 3.2: Estrutura básico de um bloco no Bitcoin.



- Versão Blockchain: Consiste na versão da estrutura do bloco.
- *Hash Anterior*: O valor do *hash* do bloco anterior, calculado usando uma função de *hash* segura (no caso do Bitcoin, SHA-256 [11]).

[1]O termo "minerador" deriva do processo conhecido no Bitcoin como *mineração*, que consiste em realizar o esforço computacional necessário para verificar os dados a serem colocados no bloco, conforme discutido mais profundamente na Subseção 3.1.2

- *Merkle Tree Root*: O valor de *hash* correspondente à raiz de uma árvore de Merkle [12] construída a partir de todas as transações incluídas neste bloco.

- *Timestamp*: Data e hora de criação do bloco.

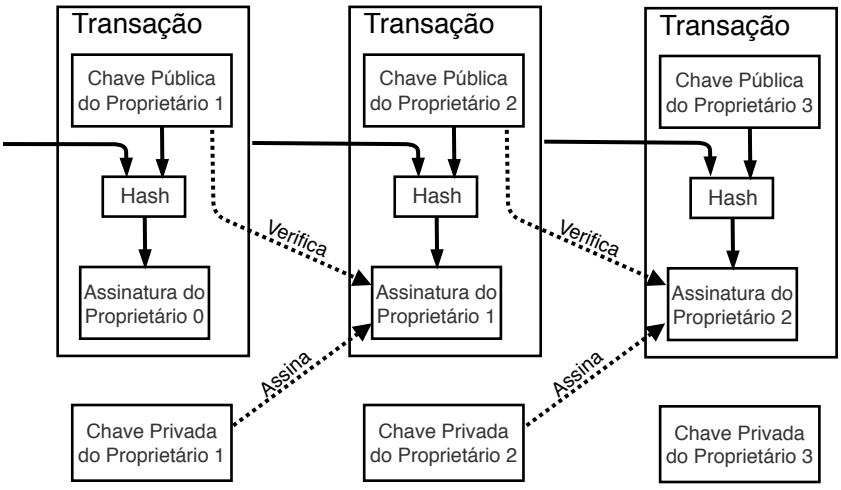
- *nBits*: Valor que representa a dificuldade computacional para mineração desse bloco. Este campo está diretamente relacionado com o mecanismo de consenso utilizado pelo Bitcoin, conhecido como *prova de trabalho* (*Proof of Work* (PoW)), conforme discutido mais adiante na Subseção 3.1.2.

- *Nonce*: Valor arbitrário adicionado ao bloco para dar variabilidade ao valor do *hash* do bloco. Como o campo "nBits", este campo também está relacionado ao mecanismo de consenso via PoW.

Quando uma nova rede de Blockchain é desenvolvida, um bloco conhecido como *Gênesis* ou Bloco Zero é criado. O Bloco Zero é um bloco diferente dos demais, pois, pelo fato de ser o primeiro bloco da cadeia, ele não possui *hash* anterior. Além disso, o Bloco Zero pode conter informações diferentes de acordo com a implementação do Blockchain (e.g., dados de configuração).

Com essa estrutura baseada em encadeamento de *hashes*, o Blockchain garante a integridade e da sua ordem relativa dos blocos. Aliado à assinatura digital dos registros armazenados, pode-se também verificar a autenticidade dos dados de forma distribuída. Em particular, quando esses dados assumem a forma de transações de ativos entre usuários, pode-se identificar o atual proprietário de um ativo qualquer simplesmente analisando a cadeia de transações válidas realizadas. Isso é ilustrado na Figura 3.3, que mostra como diferentes transações alteram o proprietário do objeto da transação (e.g., uma criptomoeda): ao assinar uma transação, o dono do ativo indica sua anuência em transmití-lo ao novo proprietário; o último usuário identificado como receptor do ativo é então seu atual proprietário.

Figura 3.3: Exemplo de aplicação do Blockchain: ordem relativa das transações revela o atual proprietário do ativo que é objeto dessas transações.



3.1.2. Mecanismo de Consenso

Todos os nós no Blockchain detêm de forma independente suas próprias cópias dos blocos que o compõem, e cada registro nele contido é verificado individualmente conforme regras definidas pela camada de aplicação [10]. Após a verificação de um bloco, o nó do sistema envia uma cópia desses blocos para outros usuários da rede, mecanismo que se repete até que todos os nós recebam a nova informação.

Como esse processo é feito de forma assíncrona e sem coordenação central, cada nó pode ter uma visão diferente do estado do Blockchain em um dado instante. Entretanto, para garantir a convergência dessas visões em um espaço de tempo não muito longo, o Blockchain utiliza um *mecanismo de consenso*. Isso é essencial para criar um sistema consistente, no qual todos os nós concordem com a ordem dos blocos e sobre os seus conteúdos. Em uma aplicação voltada a troca de ativos, por exemplo, é isso que garante que todos os usuários têm a mesma visão sobre quem é o proprietário de um determinado ativo, evitando tentativas de gasto duplo. Nesse cenário, o mecanismo de consenso permite construir um ambiente bastante resiliente a tentativas de violação, no qual as transações envolvendo qualquer ativo digital são verificadas por uma gama de participantes não mutuamente confiáveis [1].

É interessante notar que a questão do consenso não surgiu com o Blockchain. Na realidade, trata-se de um problema computacional bastante antigo na área de computação distribuída, que deve considerar garantias de acordos de maneiras democráticas, acordos estes que garantem a confiabilidade do que é acordado e distribuído [13]. Em qualquer cenário, incluindo no Blockchain, o consenso tem como função a aplicação/verificação de um conjunto regras estabelecidas por um conjunto de participantes atuando de forma organizada. Assim, a especificação de um esquema de consenso envolve, além da organização dos nós e da definição de suas regras de atuação, a própria infraestrutura sobre a qual o Blockchain opera, (e.g., a forma como as mensagens são trocadas, a organização da rede e os algoritmos empregados). Alguns dos principais subcomponentes que formam mecanismos de consenso são, portanto [2]:

- Topologia de rede de consenso. Está relacionada ao nível de descentralização no processo de validação e também a fatores como o mecanismo de recompensa utilizado. As principais topologias adotadas no Blockchain são similares às adotadas em outras redes P2P, a saber: descentralizada, na qual todos os nós têm as mesmas responsabilidades; parcialmente centralizada, em que assume-se a aparição dinâmica de "super-nós", que assumem responsabilidades adicionais na rede; hierárquica, na qual os nós se organizam de forma bastante rígida, seguindo uma relação de responsabilidades bem definida; e centralizada, que envolve alguma entidade (parcialmente) confiável, como um servidor.

- Comunicação e troca de mensagens. Os Blockchains também são sistemas de armazenamento redundantes e descentralizados. Esta redundância torna difícil sequestrar/roubar as informações armazenadas neles. Como essa informação trafega através de redes e normalmente não há uma autoridade de coordenação central, cada nó deve transmitir as informações que possui (e.g., novos blocos ou mesmo o Blockchain completo) para outros nós que sabe-se estarem participando do sistema.

Para isso, os nós possuem uma lista (não necessariamente completa) dos demais nós da rede. Sempre que um novo bloco é adicionado à versão local do Blockchain de um nó, este nó passa o bloco para outros em sua lista de nós por meio de protocolos de propagação de mensagens via *broadcast*, como é o caso de mecanismos de fofoca (*gossip*) [14]. Já em redes nas quais os nós são conhecidos, pode-se estabelecer um conjunto de nós que ficam responsáveis exatamente pelo repasse de mensagens, como acontece em soluções como Ripple [15]. Há então dois tipos de trocas de mensagens [2]:

  - Local: A troca de mensagens ocorre primeiro entre nós vizinhos, através de um processo de validação local, e posteriormente se propaga pela rede até que o consenso global seja alcançado. Esse tipo de mecanismo também é chamado de "consenso federado". Um exemplo de sistema que usa esse processo é o Ripple, no qual os nós podem compartilhar registros de transações entre si e chegar a um consenso sem conhecer diretamente todos os nós da rede.
  - Global: Esse é o tipo mais comum na maioria das implementações de Blockchain (e.g., Bitcoin, Ethereum, etc.). Neste caso, a comunicação ocorre em uma lista de nós selecionados, conhecidos na rede Bitcoin como nós de *fallback*, os quais são responsáveis por manter uma lista de todos os nós na rede. Na conexão de um novo nó, os nós de *fallback* enviam uma lista de nós escolhidos aleatoriamente para o participante. A topologia de rede resultante carece de um conceito de proximidade ou vizinhança local.

- Acordo para o consenso e resolução de conflitos. Define um conjunto de regras sob as quais os registros (como conjuntos de transações ou qualquer outra parte atômica) são atualizados independentemente pelos nós de um sistema distribuído. Isso é importante para entender como um sistema distribuído é capaz de lidar com as chamadas falhas bizantinas, i.e., falhas que podem apresentar-se de forma diferente para observadores distintos, devido à ausência de uma visão global da rede, criando então inconsistências no sistema. Dependendo do cenário, esse tipo de falha por ser tratado por meio de comunicações síncronas ou assíncronas, aliadas a um mecanismo de votação para decidir conflitos. Uma vez estabelecido um consenso na rede, ele pode ser considerado [2]: determinístico, i.e., a informação armazenada em um Blockchain após o consenso não pode ser alterada posteriormente; ou não-determinístico, nos quais o estabelecimento de um consenso em certo momento não impede que o estado do Blockchain seja alterado por um consenso posterior (embora possa reduzir a probabilidade de tais modificações). Para um cenário altamente distribuído, como é o caso da maioria dos sistemas baseados em Blockchain, o consenso determinístico é bastante difícil de alcançar. Por exemplo, no Bitcoin os novos blocos se difundem por meio de mensagens assíncronas e os nós têm por regra armazenar localmente a versão do Blockchain que apresentar o maior número de blocos (a chamada "regra da cadeia mais longa"); nesse caso, mesmo que seja alcançado um consenso global sobre o estado do Blockchain, nada impede que um conjunto de novos nós com poder computacional suficiente entre no sistema e anule o consenso anterior oferecendo versões mais longas da cadeia de blocos.

- Algoritmo para inserção de blocos. Um Blockchain, como um tipo especial de sistema distribuído, é considerado tolerante a falhas devido à replicação de seu estado por todos os nós. Assim, mesmo que alguns nós deixem a rede (e.g., devido a um desligamento acidental ou proposital), a aplicação baseada em Blockchain tem a capacidade de continuar funcionando, i.e., é capaz de manter o mesmo grau de confiabilidade e validade das informações armazenadas nos seus registros que apresentava antes da partida daquele nó. De fato, Blockchains representam uma solução descentralizada para o armazenamento de informações nos quais há elevado grau de redundância, já que cada nó do sistema possui uma réplica do livro razão. Para que haja consistência entre os dados armazenados pelos diferentes nós, é essencial que sejam definidas regras para a atualização desses dados de forma distribuída. Caso contrário, se qualquer nó puder a qualquer momento atualizar sua versão local e disseminá-la para outros nós, dificilmente haveria uma convergência entre as diversas versões do Blockchain espalhadas pela rede. Nos últimos anos, a evolução das tecnologias de Blockchain foi acompanhada pelo desenvolvimento de diferentes mecanismos de regulação da inserção de blocos no Blockchain que ajudam a manter a consistência das informações contidas no livro razão. Alguns dos principais são:

  - *Proof of Work* (PoW): Introduzido no Bitcoin [6], foi o primeiro e ainda é provavelmente o mais conhecido mecanismo de consenso distribuído no cenário do Blockchain. No PoW todos computadores da rede são encarregados de manter a segurança do Blockchain, e comumente o fazem em troca de gratificações. Tecnicamente, a tarefa desempenhada pelos mineradores consiste em encontrar uma entrada cujo valor de *hash* seja menor que um determinado alvo. Essa é uma tarefa computacional extremamente repetitiva e que envolve um alto custo computacional, de modo que comumente é realizada por meio de hardware especializado. Apesar desse mecanismo de consenso fazer com que seja custoso adicionar novos blocos no Blockchain, é simples a verificação dos blocos inseridos (basta o cálculo de uma única função de *hash*), e o esquema é bastante eficiente para combater mineradores maliciosos.
  - *Practical Byzantine Fault Tolerance* (PBFT): Trata-se de um algoritmo de replicação originalmente criado para permitir a qualquer sistema tolerar falhas bizantinas. Os nós se organizam para operar em rodadas, de modo que em cada rodada um nó primário é selecionado de acordo com certas regras. O nó primário fica então responsável por inserir o próximo bloco na cadeia. O processo é dividido em três fases: Pré-Preparado, Preparado e Comprometido. Para passar de uma fase a outra, um nó precisa receber o voto de 2/3 de todos os nós. Para que isso seja possível, portanto, exige-se que o número total de nós seja conhecidos pela rede. Não há mecanismos custosos computacionalmente nesse caso, bastando a consulta entre nós para se chegar a um consenso [16].
  - *Proof of Authority* (PoA): Os participantes não são solicitados a resolver problemas matemáticos arbitrariamente difíceis, mas devem usar um conjunto de autoridades configuradas para colaborar sem confiança. Como essas autoridades são nós com permissão especial para criar novos blocos e proteger o

Blockchain, normalmente os mecanismos PoA não são utilizados em redes públicas, nas quais tal confiança não existe. Por outro lado, esse mecanismo se encaixa bem em redes privadas do Modelo Consórcio, nas quais algumas entidades reais pré-selecionadas atuam como autoridades. Para prevenir a personificação desses nós especiais por outros nós, as autoridades recebem certificados digitais válidos para assinar os novos blocos. Assim, cada bloco (ou cabeçalho) que um cliente vê pode ser comparado com uma lista de signatários confiáveis [2].

- *Proof of Burn* (PoB): Os mineradores devem provar que "queimaram" alguns ativos digitais (e.g., criptomoedas). Eles fazem isso enviando os ativos em questão para um endereço especial do Blockchain, ao qual não é associada a capacidade de repassar esses ativos. Embora o PoB tenha sido criado com o objetivo de minimizar o desperdício de recursos gerados pelo PoW, atualmente todos os mecanismos do PoB funcionam exatamente pela queima de moedas digitais mineiradas pelo PoW. Portanto, o mecanismo acaba sendo caro, tendo em vista que as moedas digitais usadas como combustível para "queima" em um sistema PoB não podem ser recuperadas [17].
- *Proof of Capacity* (PoC): também conhecida como *Proof of Space*, *PoSpace* ou *PoStorage*, esse mecanismo é baseado no conceito de espaço como recurso (*Storage as a Resource* – SaaR). O foco desse mecanismo não está nos ciclos da CPU, mas na quantidade de armazenamento real não volátil (e.g., disco rígido e SSD) que o minerador deve empregar para ganhar o direito de inserir um novo bloco no Blockchain. Em outras palavras, os nós devem alocar um volume significativo de espaço em memória secundária para a mineração, em vez de usar a CPU como no PoW. Mineradores que dedicam mais espaço de armazenamento têm uma expectativa proporcionalmente maior de minerar com sucesso um bloco e receber a recompensa correspondente [18, 19].
- *Proof of Stake* (PoS): É uma abordagem alternativa ao PoW que busca reduzir o custo de energia do processo de mineração, bem como a resultante dependência de hardware especializado para fazê-lo. Basicamente, esse tipo de mecanismo dá preferência a mineradores que tenham maior participação na rede e, portanto, tenham mais a perder no caso de fraudes. Por exemplo, em um sistema de criptomoedas, seria dada preferência aos mineradores que detenham mais moedas no momento da inserção do bloco [19].
- *Proof of Importance* (PoI): Implementado pela companhia NEM, este tipo de mecanismo leva em consideração a importância dada um minerador na rede, com base no número de moedas que ele possui, no número de transações por ele realizadas, e na sua reputação dentro da rede [19]. Esse tipo de mecanismo pode, portanto, ser visto como uma extensão do conceito de PoS.
- *Delegated Proof of Stake* (DPoS): Este mecanismo basicamente elege uma lista de nós que terão a oportunidade de participar do bloco de novas transações e adicioná-los ao Blockchain. A prova de participação delegada tem a função de incluir todos os detentores de moedas, mesmo não recompensando da mesma maneira que a PoS. A ideia central é dar maior responsabilidade e importância para os titulares da rede [19].

- *Leased Proof of Stake* (LPoS): Uma variante do PoS que fornece uma possibilidade de mineradores pequenos gerarem lucros. Basicamente, este mecanismo permite que os nós com maior participação na rede (e, portanto, maior chance de validarem blocos) sejam alugados por outro nós [19].
- *Ripple*: O Ripple utiliza um protocolo próprio, denominado XRP, para que opere em sub-redes confiáveis coletivamente dentro da rede maior. Na rede, os nós são divididos em dois tipos: Servidor, que participa do processo de consenso; e cliente, que apenas transfere. Na rede Ripple, os servidores de validação trocam informação sobre os registros que precisam ser incluídos no Blockchain; o sub-conjunto de transações inserido no próximo bloco corresponde àquelas que receberam o voto positivo de uma quantidade mínima (e ajustável) de validadores. Para facilitar o consenso, cada validador pode definir um conjunto de outros validadores nos quais ele confia, de modo que seus votos dão preferência às transações propostas por nós em sua rede de confiança [15].
- *Tendermint*: Semelhante ao PBFT, esse mecanismo segue o mesmo padrão de segurança. A diferença é que no Tendermint exige que os validadores bloqueiem suas moedas enquanto participam do processo de validação, e tentativas de fraudar o sistema (e.g., criar situações de gasto duplo) são punidas pelos nós da rede (e.g., com a perda de moedas) [20].

As características de cada mecanismo de consenso devem ser considerados em conjunto ao projetar um processo ativo de validação de consenso de rede, porque não apenas sua configuração individual, mas também sua combinação determinam quando e como o consenso no Blockchain é alcançado e o livro razão é atualizado. A Tabela 3.1 mostra uma comparação entre alguns dos diferentes mecanismos de consenso, classificados em três funcionalidades: Gerenciamento do nó, Economia de Energia e Controle do Nó.

Tabela 3.1: Tabela de Comparação entre mecanismos de consenso Blockchain.

Funcionalidades	**Gerenciamento do Nó**	**Economia de Energia**	**Controle do Nó**
PoW	Aberto	Não	25% do poder *hash*
PoS	Aberto	Parcial	51% *stake*
DPoS	Aberto	Parcial	51% mineradores
LPoS	Aberto	Parcial	51% *stake*
PBFT	Permissão	Sim	33.3% de falhas
PoI	Aberto	Sim	51% mineradores
PoA	Permissão	Parcial	51% mineradores
PoB	Aberto	Parcial	51% mineradores
PoC	Aberto	Sim	51% *Stake*
Ripple	Aberto	Sim	20% falhas UNL
Tendermint	Permissão	Sim	33.3% poder de voto

O *Gerenciamento do nó* refere-se a quem pode participar do mecanismos de consenso, podendo ser aberto a qualquer pessoa ou necessitar de autorização para fazer parte

do Blockchain. A *Economia de energia* informa se a concepção do mecanismo de consenso tem em suas finalidades economizar energia, aspecto que tem ganhado importância em particular depois que o custo de mineração via PoW do Bitcoin começou a elevar-se demasiadamente. Por fim, o *Controle do nó* refere-se às condições para que o processo de consenso possa ser comprometido de algum modo indevido (e.g., manipulação, mineração egoísta, etc.).

Diferentes mecanismos de consenso são apresentados na Tabela 3.1 e para cada um destes há diferentes contextos de aplicação, benefícios e potenciais dificuldades. Para cada versão de Blockchain existem diferentes métodos de aplicação destes mecanismos. Por exemplo, embora ambas sejam plataformas de gestão de contratos inteligentes, a *Bitshares* adota DPoS, enquanto o *Ethereum* utiliza PoS. De forma geral, os mecanismos PoW e PoS são mais os mais utilizados para Blockchain do tipo Público, enquanto Blockchains de Consórcio e Privado têm preferência pelos mecanismos PBFT e DPoS, que consomem menos energia, ou então PoW devido ao grau de segurança resultante.

3.1.3. Taxonomia de Modelos Blockchain

Atualmente, os sistemas que implementam o conceito de Blockchain podem ser classificados em três modelos principais quanto ao seu acesso [5, 21]:

- Blockchain Público: é o modelo mais conhecido. Nesse tipo de Blockchain, qualquer pessoa pode ler, enviar, ou validar transações, além e também poder participar do processo de consenso distribuído. Esses Blockchains são considerados totalmente descentralizados, pois a influência que uma pessoa tem no processo de consenso depende apenas de seus méritos em comparação com outros nós na rede (e.g., no caso de Blockchains baseadas em PoW, sua capacidade de mineração é proporcional ao seu poder computacional). Assim, pode-se dizer que o processo de consenso em Blockchains públicas é bastante democrático.

- Blockchain Consórcio: composto por duas ou mais instituições. Este modelo é basicamente controlado e modelado por instituições parceiras, que podem alterar as regras de acesso ao Blockchain conforme seus interesses e necessidades. Este modelo é considerado parcialmente descentralizado, pois não é controlado por apenas uma parte, mas sim por um conjunto pré-definido de instituições.

- Blockchain Privado: é um modelo mais conservador, no qual apenas uma instituição detém o controle sobre as operações. Seu modelo de funcionamento pode mudar de acordo com a necessidade e interesse desta instituição, de forma monocrática. Esse tipo de Blockchain é considerado centralizado e de forma geral é utilizado para auditoria, gerenciamento de banco de dados e outras informações que necessitam de um nível diferenciado de controle sobre os dados armazenados.

A Tabela 3.2 lista alguns critérios de comparação entre os modelos de Blockchain: Público, Consórcio e Privado. Estes modelos são avaliados de diferentes formas, baseados nos conceitos da tecnologia Blockchain [22]:

- Consenso distribuído: Se todos os nós podem participar do processo de consenso, ou apenas nós pré-determinados.

- Permissão de Verificação: Se as transações podem ser verificadas de modo público ou restritamente.
- Imutabilidade: O Blockchain, por princípio, cria uma estrutura de dados imutável após um consenso amplo ser atingido. Contudo, algumas implementações podem possibilitar alterações nos dados armazenados, que geralmente ocorrem somente por processos de consenso distribuído adicionais ou por meio da atuação de uma entidade confiável no sistema.
- Centralização: refere-se ao grau de controle de alguma instituição sobre o Blockchain.
- Processo de consenso: Se qualquer entidade pode participar do processo de consenso ou apenas entidades certificadas ou pré-selecionadas.

Tabela 3.2: Tabela de Comparação modelos de acesso do Blockchain.

	Blockchain Público	**Blockchain Consórcio**	**Blockchain Privado**
Consenso Distribuído	Todos Mineradores	Mineradores Selecionados	Mineradores Selecionados
Permissão de Verificação	Pública	Restrita	Restrita
Imutabilidade	Sim	Adulterável	Adulterável
Centralização	Descentralizado	Parcial	Centralizado
Processo de Consenso	Todos Mineradores	Mineradores Selecionados	Mineradores Selecionados

3.1.4. Versões Blockchain

Embora os mecanismos subjacentes ao Blockchain existam há muito tempo na literatura, pode-se afirmar que a tecnologia de Blockchain como utilizada atualmente é razoavelmente recente (surgiu em 2008 com o Bitcoin). Esse curto período de existência não impediu, entretanto, que os conceitos por trás da tecnologia evoluíssem. De fato, alguns autores identificam diferentes eras ou estágios de evolução do Blockchain, que, embora nem sempre tenham sido formalmente definidos, apresentam algumas características marcantes que os diferenciam [1]. Essas características são discutidas nas próximas seções.

3.1.4.1. Blockchain 1.0

A primeira versão do Blockchain muitas vezes é confundida com a própria aplicação que faz uso dessa tecnologia, o Bitcoin. O Bitcoin é uma solução para um problema antigo com dinheiro digital, que é o gasto duplo. Até o surgimento do Bitcoin, soluções envolvendo dinheiro digital tratavam esse tipo de dado como um ativo digital comum, que poderia ser copiado diversas vezes como qualquer arquivo ou mensagem. Portanto, sem as devidas precauções, transações sobre um mesmo ativo financeiro poderiam ser autorizadas sem a devida verificação prévia de sua existência ou de seu proprietário. A solução praticada para resolver esse problema consistia basicamente na delegação de confiança a uma terceira parte (e.g., bancos, Paypal, etc.), que ofertava o serviço de controle do ativo financeiro. Esse terceiro confiável seria responsável por garantir as identidades das

partes envolvidas na transação e o cumprimento das regras nela envolvidas (e.g., o ativo seria transferido após a conclusão da transação), além de prevenir a ocorrência gastos duplos [1].

Nesse cenário, o Blockchain 1.0 moldou-se como uma moeda para a Internet, criando o sistema de pagamento digital que é a primeira aplicação do Blockchain. A principal funcionalidade das moedas baseadas em Blockchain é de que qualquer transação pode ser realizada sem a necessidade da confiança entre as partes envolvidas na transação, de modo completamente descentralizado, distribuído e global. Esta habilidade cria possibilidades muito além de moedas e pagamentos, e que são aproveitadas no Blockchain 2.0.

3.1.4.2. Blockchain 2.0

Após o uso inicial como solução para dinheiro digital, o Blockchain foi visto como uma tecnologia que pode ser explorada em outros setores da tecnologia. Durante conversas entre os desenvolvedores originais do Bitcoin, foi indicado que o Blockchain suportaria uma variedade de tipos de transações possíveis, não apenas criptomoedas. Estas transações podem ser, por exemplo, contratos alfandegários, arbitragens de terceiros, transações de custódia e outros tipos de modelos.

É interessante notar que essa expansão no uso da tecnologia Blockchain foi motivada especialmente pela elevada escala assumida pelo Bitcoin, e pela robustez observada no seu mecanismo de consenso a despeito dos custos energéticos envolvidos [1]. Outro fator importante foi o modelo baseado em incentivos para os usuários que contribuem com a rede (os mineradores), algo essencial em qualquer tecnologia P2P. Por outro lado, conforme previamente mencionado, muitas aplicações são baseadas em uma concepção equivocada sobre (1) quais serviços de fato são fornecidos por um Blockchain, (2) quais serviços poderiam ser obtidos usando simplesmente os mecanismos subjacentes a ele (e.g., assinaturas digitais), e (3) quais serviços não são fornecidos. Por exemplo, não é incomum que aplicações nas quais é necessário verificar apenas a *existência de registros*, mas não importa sua ordem relativa, utilize um Blockchain quando a simples assinatura digital desses registros bastaria. Em outros casos, assume-se que o mecanismo de consenso utilizado no Blockchain é suficiente para decidir sobre a validade dos dados registrados, quando na realidade o consenso apenas decide sobre a *ordem relativa* dos registros, deixando a definição das regras de validação dos dados a cargo da camada de aplicação construída sobre o Blockchain.

De qualquer forma, o Blockchain 2.0 foi elaborado tomando proveito do conceito de transação descentralizadas subjacentes à tecnologia, podendo ser usado para registrar, confirmar e transferir todos os tipos de contratos e patrimônio. Sem entrar no mérito da real aplicabilidade de um sistema de Blockchain a cada aplicação específica, alguns exemplos de contratos e patrimônios suportados pelo Blockchain 2.0 são [1]:

- Geral: Garantia de transações, contratos coligados, arbitragem por terceiros, aplicação de multas.
- Transações Financeiras: estoque, captação própria, financiamento colaborativo,

cartas de confiança, fundos mútuos, pensões, derivados, anuidades.

- Registros públicos: escritura de terra e patrimônio, registro de veículos, licença de comércio, certidão de casamento, certidão de óbito.
- Identificações: carteira de motorista, carteira de identidade, passaporte, título eleitoral.
- Registros privados: empréstimos, contratos, apostas, assinaturas, garantias, testamento.
- Atestados: casa, quartos de hotel, carros alugados, acesso automotivo.
- Chaves de ativos físicos: casa, quartos de hotel, carros alugados, acesso automotivo.
- Ativos intangíveis: patente, marca comercial, direito autoral, nomes de domínio.

3.1.4.3. Blockchain 3.0 e além

Enquanto o uso inicial do Blockchain se concentrava no registro de ativos financeiros (criptomoedas) e soluções seguintes foram voltadas ao registro de ativos diversos, as propostas atuais costumam ter como foco as aplicações que podem ser construídas com base nesses registros. A maioria dessas aplicações é voltada à construção de sistemas altamente descentralizados, fornecendo um bom grau de verificabilidade mesmo na ausência de entidades confiáveis (ou, mais precisamente, buscando substituir tais entidades).

Um exemplo nesse sentido é o *Namecoin: Domain Name Systen (DNS) Descentralizado* [23]. Lançado entre 2011, o Namecoin usa um sistema de monetização de trabalho computacional similar ao do Bitcoin, mas em vez de criar uma moeda "genérica", seu objetivo primário é servir como um repositório para permitir a verificação de registros DNS. A aplicação é um uso não-usual da tecnologia Blockchain para uma aplicação de alcance e interesse globais, mas que diferentemente da Internet tradicional não pode ser controlada por qualquer corporação ou governo. Assim, de modo similar a outros sistemas de armazenamento de dados P2P (e.g., Freenet [24]) o *Namecoin* visa criar um sistema resistente à censura ou repressão, no qual usuários consigam publicar informações livremente na Internet. Essa é uma das motivações pelas quais o *InterPlanetary File System* (IPFS) [25], um sistema de armazenamento descentralizado de conteúdo Web sem a necessidade de um servidor de hospedagem, que propõe o uso de do *Namecoin* como uma das formas de aumentar seu grau de descentralização.

Outro exemplo interessante é o BitID [26], que fornece um serviço de verificação de identidade de indivíduos com base no seu identificador no Blockchain do Bitcoin.

Embora emblemáticos, esses exemplos estão longe de dar a dimensão completa do que é atualmente classificado como Blockchain 3.0. Afinal, esta fase é marcada por criar um ambiente bastante rico de aplicações distribuídas, para propósitos como proteção de propriedade intelectual, rastreamento de cadeia de suprimentos/*supply chain*, pagamentos internacionais, internet das coisas, e privacidade de pacientes no tratamento médico [5].

Ironicamente, embora várias propostas nessas áreas se definam como parte do "Blockchain 3.0", muitas delas o fazem apenas por serem baseadas em redes *Peer-to-Peer* (P2P), embora não envolvam de fato princípios fundamentais que caracterizam o Blockchain original, como encadeamento de blocos ou mecanismos de consenso. Independentemente de eventuais discussões sobre uso indevido do termo "Blockchain" nesses casos, o fato é que tais esforços têm levado a uma grande expansão no número e na variedade de aplicações descentralizadas na Internet. Exatamente por isso, um requisito importante para a evolução das tecnologias do Blockchain 3.0 é a integração efetiva não só entre diversas plataformas, mas também na indústria com ferramentas e tecnologias existentes. Por exemplo, propostas como Polkadot [27] e Aelf [28] têm como objetivo é realizar a integração entre diversas Blockchains de fins específicos. Os desafios envolvidos envolvem questões de padronização entre interfaces de comunicação e estruturas de dados, fatores determinantes para a interoperabilidade entre soluções.

3.1.4.4. Comparação entre as versões do Blockchain

Com a apresentação das três versões do Blockchain, se faz necessário um comparativo destas versões da tecnologia. A Tabela 3.3 realiza esta comparação apresentando: funcionalidades, abrangência de consenso distribuído e suas principais aplicações.

Tabela 3.3: Comparativo básico entre as versões do Blockchain.

	Blockchain 1.0	**Blockchain 2.0**	**Blockchain 3.0**
Funcionalidades	Criptomoedas	*Smart Contracts*	Aplicativos Descentralizados
PoW	Sim	Sim	Sim
DPoS	Não	Sim	Sim
PoS	Sim	Sim	Sim
PBFT	Não	Sim	Sim
LPoS	Não	Sim	Sim
PoI	Não	Sim	Sim
PoA	Não	Sim	Sim
PoB	Sim	Sim	Sim
PoC	Não	Sim	Sim
Ripple	Não	Sim	Sim
Tendermint	Não	Sim	Sim
Aplicações	Bitcoin	Ethereum,MasterCoin, *Open assets*, *Colored Coins*	Computação Descentralizada, Armazenamento Descentralizado

Observa-se a partir da Tabela 3.3 a evolução de tecnologias baseadas em Blockchain através de suas diferentes versões. Inicialmente, partiu-se de uma solução para o problema de centralização das instituições financeiras, evoluindo para o desenvolvimento e utilização de contratos inteligentes, e por fim para aplicações de computação e armazenamento descentralizados, que envolve diferentes setores da indústria.

Outro ponto visível, é que foi percebido o elevado gasto energético para a mineração dos blocos de Blockchain, além de sua lentidão e vazão, a partir deste pretexto foram desenvolvidos outros mecanismos além do PoW [29]. O mecanismo de consenso DPoS basicamente aproveita o poder da votação de aprovação de partes interessadas delegadas para resolver problemas de consenso e validar o Blockchain em um modelo com designs semelhantes aos sistemas democráticos, consequentemente utiliza um menor gasto energético que o o modelo PoW e os usuários votam proporcionalmente ao que produzem [30].

Por fim, percebe-se que dentre todos mecanismos que são propostos cada qual tem seu benefício em comparação um com os outros, mas o PoW é o mecanismo que, apesar de ter um custo elevado, é o que mais se encaixa no quesito de segurança e procedência na aplicação em conjunto com outras tecnologias.

3.2. Segurança e Aplicações de Blockchain

Não há dúvidas quanto a popularidade da tecnologia Blockchain, mais do que isso o Blockchain tem feito um impacto duradouro no mundo [31]. Como exemplo de seu desenvolvimento, as características do Blockchain tornam sua utilização uma ideia atrativa em muitas áreas de negócios como setores financeiros, governamentais, industriais, farmacêuticos, saúde e segurança cibernética.

Com o crescimento do escopo de aplicação da tecnologia Blockchain e a acessibilidade aos modelos do Blockchain, novas plataformas baseadas na tecnologiaBlockchain foram desenvolvidas. Estas plataformas fornecem suporte para diversas aplicações, mecanismos de consenso, modelos de Blockchain e outras características.

A partir da versão 3.0 do Blockchain, a tecnologia expandiu-se para aplicações e armazenamento descentralizados, a qual beneficia os modelos de Blockchain de consórcio e privado. Sistemas de Blockchain como Linux Foundation's Hyperledger [8] e Multichain [9] foram desenvolvidos propriamente para estes dois modelos, mas sistemas como Credits e Ethereum, possuem versões aplicáveis nos três modelos Blockchain.

Como apresentado (Subseção 3.1.3) o modelo mais utilizado de acesso é o Público, sendo este o que possui maior escopo de plataformas Blockchain desenvolvidas, como Bitcoin [32], Ripple [15], Ethereum [7], Credits [33], entre outros. Para este minicurso são descritas três plataformas que possuem em seus modelos o Blockchain Privado/Consórcio. As plataformas abordadas são: Ethereum, Hyperledger e MultiChain.

3.2.1. Ethereum

O Ethereum [34] oferece um protocolo alternativo para criação de aplicativos descentralizados, fornecendo um conjunto diferente de compensações/recompensas que sejam úteis para uma classe considerável de aplicativos descentralizados. O foco da tecnologia leva em consideração o suporte para o desenvolvimento da aplicação, assim como segurança e a capacidade de diferentes aplicações interagirem de forma muito eficientes.

O Ethereum é um Blockchain de propósito geral que suporta a execução de contratos inteligentes por meio de uma máquina virtual chamada *Ethereum Virtual Machine* (EVM). A EVM é uma camada de abstração, executando no *host* que conecta o cliente com a rede e oferece o ambiente que executa as instruções dos contratos inteligentes de terceiros em uma rede global de computadores. Uma vez criados, os contratos são compilados na EVM que gera o bytecode para o hardware do hospedeiro. Neste modelo, cada operação realizada por um contrato está associado com um custo de execução, fazendo com que transações que modifiquem o estado de um contrato exijam um combustível (chamado de *gas*) ou incentivo financeiro associado com a transação.

A tecnologia empregada no Ethereum possibilita a fácil escrita de contratos inteligentes e construção de aplicativos descentralizados, nos quais possam os usuários possam

criar as suas próprias regras arbitrárias [35]. A partir do desenvolvimento da plataforma Ethereum foi possível, de forma real, verificar as diversas possibilidades de uso da tecnologia Blockchain que até aquele ponto havia apenas sido utilizada com finalidades do setor financeiro.

3.2.1.1. Contas Ethereum

O Ethereum é composto por objetos(contas), que contém um endereço de vinte bytes e transições de estado, sendo transferências diretas de valor e informação entre objetos. Um objeto Ethereum possui quatro campos:

- *Nonce* é um contador usado para garantir que cada uma destas transações possam ser processadas apenas uma vez;
- Balanço da conta.
- Código do contrato, se existir.
- Armazenamento da conta.

O Ethereum possui sua própria forma para financiar a mineração e monetizar a mesma, o seu "combustível", como é definido pela plataforma, é conhecido como *Ether*. De forma geral o Ethereum possui dois tipos de conta:

- Propriedade externa que é controlada por chaves privadas.
- Contas de contrato que são controladas por seu código de contrato.

As contas do Blockchain realizam o processo de comunicação, troca de mensagens e de transações. Estas contas do Ethereum podem ser somente usuários ou também mineradores que obtém seus ganhos a partir da moeda.

3.2.1.2. Mensagens e Transações

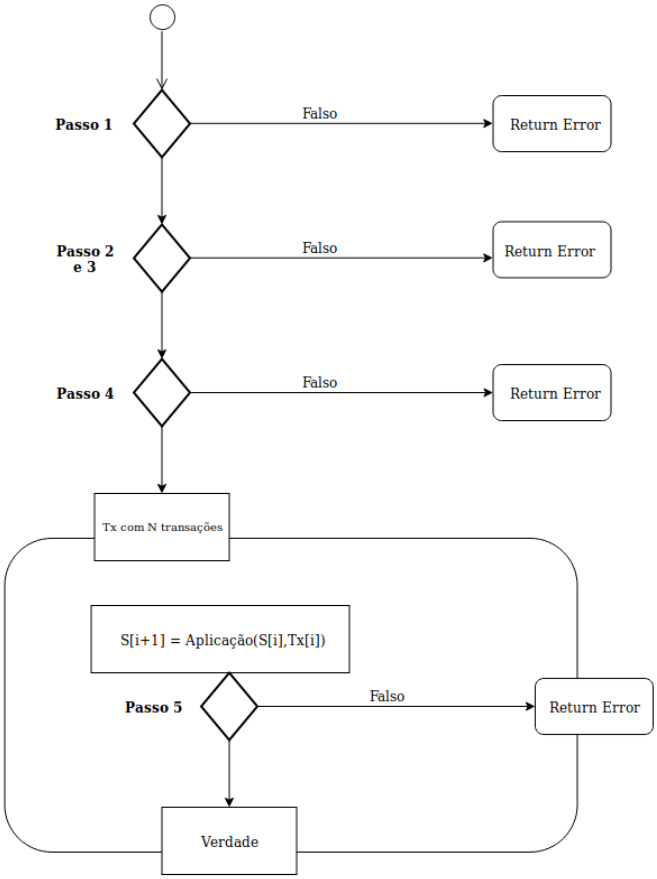
As transações são pacotes de dados assinados que armazenam uma mensagem a ser enviada de uma conta de propriedade externa. As transações possuem: Destinatário, assinatura de identificação do remetente, quantidade de *Ether* para o destinatário, campo de dados opcional, o *STARTGAS* e um *GASPRICE*, o último representa a taxa que o remetente paga por etapas computacionais [35].

Os campos *STARGAS* e *GASPRICE* são cruciais para o modelo de negação de serviço do Ethereum. O uso destes campos tem como finalidade evitar a ocorrência de *loops* infinitos, assim como desperdícios computacionais que são desnecessários [35].

3.2.1.3. Blockchain e Mineração

O Ethereum é muito similar ao Bitcoin. Os blocos de Ethereum armazenam a lista de transações e o estado mais recente, estes mesmos blocos também armazenam o número do bloco e a dificuldade do mesmo. O fluxo básico de validação de bloco segue os seguintes passos listados na Figura 3.4 [35].

Figura 3.4: Fluxo de transição de estados no Ethereum.



Adaptado de [35]

A partir da Figura 3.4 é possível observar o fluxo de transição da validação de um bloco no Ethereum. Nesta mesma figura, percebe-se que há a necessidade de verificar a validação de todos os passos para seguir até o processo que valida o bloco.

Todos os mineradores que participam de forma ativa como mineradores, necessitam a validação de todos estes estados. A partir do processo de validação destes passos, o bloco é integrado a rede e passa a ser validado por outros nós que verificam a autenticidade do bloco e de seu minerador, permitindo que a rede seja mais segura e confiável.

3.2.2. Hyperledger

O projeto Hyperledger é um projeto de código aberto Blockchain e ferramentas relacionadas, teve seu desenvolvimento iniciado em 2015 pela Linux Foundation [8]. O Hyperled-

ger tem como visão que a tecnologia Blockchain tem potencial de impactar quase todas as áreas com atividades. Tendo em vista essa visão, acreditam que no futuro em vez de grandes Blockchains funcionando entre empresas, haverá muitos Blockchains interconectados entre si, cada um destes ajustado e adaptado para determinado propósito.

Este projeto tem como característica abranger uma considerável parte dos espectros de casos de uso, lidando com essa diversidade. Os requisitos básicos que todos os projeto da Hyperledger devem ter são:

- Transações Privadas e Contratos Confidenciais: A estrutura do Hyperledger atende aos requisitos de confidencialidade básica para algoritmos sofisticados e complexos de privacidade.
- Identidade e Auditoria: Diferentemente do Blockchain em si, o Hyperledger oferece aos usuários a capacidade de mascarar sua identidade em determinadas situações e prová-la somente quando necessário.
- Interoperabilidade: Relacionamentos com diferentes redes Blockchain.
- Modular: Estrutura extensíveis e modulares com blocos de construção comuns que podem ser reutilizados. O *WG* define módulos funcionais e interfaces para problemas como comunicação, consenso, criptografia, identidade, armazenamento do livro-razão, contratos inteligentes e política.

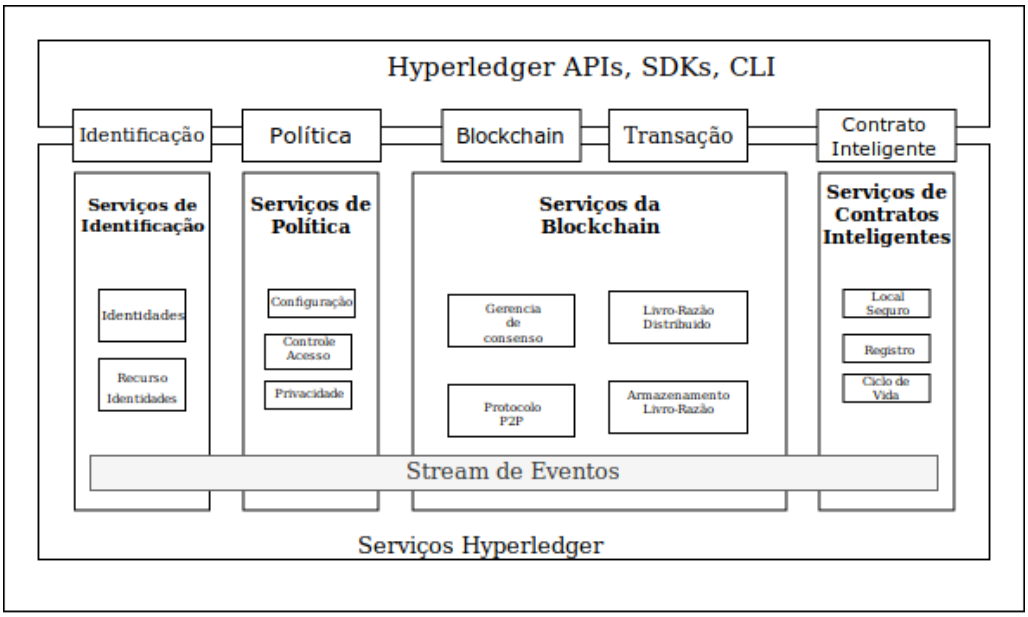
As características da tecnologia Hyperledger realizam a padronização da forma de implementação da tecnologia Blockchain, essa padronização possibilita o amplo desenvolvimento de aplicações com sua tecnologia. A arquitetura do Hyperledger garante a eficiência aos desenvolvedores, a prova deste detalhe é a quantidade de *frameworks* que foram desenvolvidos por empresas que estão disponíveis para uso de modo aberto.

3.2.2.1. Arquitetura Hyperledger

A referência de arquitetura do Hyperledger é dividida em quatro categorias que são ilustradas na Figura 3.5:

- Serviços de Identificação: Este serviço gerencia a identificação de todos participantes e componentes do sistema.
- Serviços de Políticas: Este serviço inclui permissões de acesso e autorização, incluindo políticas de confidencialidade e de consenso da implementação.
- Serviços Blockchain: Este serviço consiste no P2P, distribuição/armazenamento do livro razão e controle do algoritmo de consenso.
- Serviços de Contrato Inteligente: Este serviço inclui ambiente de tempo de execução seguro, registro de contrato inteligente e gerenciamento do ciclo de vida.

Figura 3.5: Arquitetura Hyperledger.



Adaptado de [8]

A arquitetura da especificação do protocolo Hyperledger (Figura 3.5) tem como benefício o suporte de modularidade, interoperabilidade *plug-and-play* e permite o uso de suporte de tecnologias como contêineres para suportar contratos inteligentes. Cada *framework*, desenvolvido na plataforma, possui suas finalidades e sua arquitetura é alterada de acordo com a necessidade da tecnologia.

3.2.2.2. *Framework* e Ferramentas Hyperledger

A plataforma Hyperledger inclui e promove uma variedade de tecnologias de Blockchain de negócios, é integralmente voltada para a versão Blockchain 3.0. Esta estratégia é voltada ao incentivo e reutilização de blocos de construção comuns, permitindo uma rápida inovação de componentes e promove a interoperabilidade entre os projetos.

Os principais frameworks do Hyperledger são:

- *Fabric*: Uma plataforma para criar soluções de contabilidade distribuída com uma arquitetura modular que oferece um alto grau de confidencialidade, flexibilidade, resiliência e escalabilidade.
- *Burrow*: Um cliente Blockchain modular com um intérprete de contrato inteligente com permissão desenvolvido em parte para as especificações da *máquina virtual* (VM) Ethereum.
- *Indy*: Um *ledger* distribuído que fornece ferramentas, bibliotecas e componentes reutilizáveis criados para identidade descentralizada.
- *Iroha*: Um *framework* Blockchain projetado para ser simples e fácil de incorporar em projetos de infraestrutura corporativa.

- *Sawtooth*: Uma plataforma modular para criar, implantar e executar registros (*ledger*) distribuídos.

O Hyperledger possui ferramentas e bibliotecas de utilitários para garantir esta variedade de tecnologias. As principais ferramentas disponíveis são:

- *Caliper*: Uma ferramenta de *benchmark* Blockchain que mede o desempenho de qualquer Blockchain usando um conjunto de casos de uso predefinidos.
- *Cello*: Um conjunto de ferramentas para levar o modelo de implantação sob demanda ao ecossistema Blockchain com maneiras automatizadas de provisionar e gerenciar operações Blockchain que reduzem o esforço.
- *Composer*: Um conjunto de ferramentas e estrutura de desenvolvimento aberto para facilitar o desenvolvimento de aplicativos Blockchain.
- *Explorer*: Um painel para visualizar informações na rede, incluindo blocos, *logs* de nós, estatísticas, contratos inteligentes e transações.
- *Quilt*: Um conjunto de ferramentas que oferecem interoperabilidade implementando o *Interledger Protocol* (ILP), que é basicamente um protocolo de pagamentos projetado para transferir valor em regitros/*ledgers* distribuídos e não distribuídos.

Os *frameworks* e ferramentas disponíveis da plataforma, facilitam o embasamento e utilização da tecnologia Blockchain por diferentes modelos de indústria. Desta forma, revela-se a versatilidade do uso da tecnologia Blockchain e também das possibilidades de desenvolvimento a partir do Hyperledger.

3.2.3. MultiChain

O MultiChain [9] é uma plataforma desenvolvida em 2014 para a criação e implementação de Blockchains privados, dentro ou entre organizações. A tecnologia suporta servidores MS-Windows, GNU/Linux e Apple MacOS, sendo que o MultiChain fornece uma *Application Programming Interface* (API) simples e uma interface de linha de comando.

O MultiChain resolve problemas relacionados à mineração, privacidade e abertura por meio do gerenciamento integrado de permissões de usuários. Com o uso do Blockchain privado, problemas relacionados a escala são facilmente reduzidos, além de que o Blockchain conterá apenas transações que sejam de interesse para este grupo. Três objetivos principais do MultiChain [9]:

- Assegurar que a atividade do Blockchain só seja visível para os participantes escolhidos.
- Introduzir controles sobre quais transações são permitidas.
- Permitir que a mineração ocorra com segurança sem PoW e sem custos adicionais.

Estes três objetivos principais do MultiChain são responsáveis pelo diferencial da plataforma. Um aspecto que merece destaque é que o MuiltiChain possui um administrador, sendo este responsável por toda a parte organizacional desta rede Blockchain que usa o Modelo Privado. O administrador é responsável pela criação do Bloco Gênesis, definição dos participantes da rede e também das operações que podem ser realizadas por estes. Basicamente, a rede MultiChain é constituída por um administrador, mineradores e participantes que interagem com a rede.

3.2.3.1. Mineração MultiChain

A mineração de dados no MultiChain, por ser uma plataforma de Blockchain privada resolve o problema que os participantes podem monopolizar o processo de mineração. A solução apontada pela plataforma está na restrição de número de blocos que podem ser criados pelo mesmo minerador dentro de uma janela de tempo. Este modelo de verificação impõe um tipo de *round-robin*, em que todos os mineradores autorizados devem criar blocos no formato de prioridade/rotação para gerar um Blockchain válido. A validação de um bloco MultiChain é verificada da seguinte forma [9]:

1. Aplicar todas as mudanças de permissões definidas pelas transações no bloco em ordem.
2. Contar o número de mineradores permitidos definidos após a aplicação dessa alteração.
3. Multiplicar os mineradores pela diversidade de mineração, arredondando para obter o *spacing*.
4. Se o minerador deste bloco extraiu um dos blocos anteriores de *spacing*-1, o bloco é invalidado.

A mineração, por ser privada, é restrita para certas entidades, levando ao questionamento dos seus benefícios ao utilizar este Blockchain sobre um banco de dados centralizado que aceita transações de entrada, resolve disputas e consultas relacionadas ao estado do banco de dados. Segundo a equipe do MultiChain, a utilização da plataforma se faz viável pelos seguintes ganhos [36]:

- Cada um dos participantes mantém o controle total sobre seus ativos por meio de sua chave privada.
- O controle dos registrso é distribuído por várias entidades, de modo que nenhum indivíduo ou grupo possa decidir quais transações são válidas ou confirmadas.
- Maior robustez, uma vez que problemas em um servidor não afetará o processo contínuo de transações pela rede como um todo.

Basicamente, cada membro que recebe autorização para participar da rede do MultiChain é responsável por suas ações e controles dentro da rede. Há uma comunicação entre todos os nós, que a partir do momento no qual um bloco ou uma transação é validado,

estes nós passam a realizar o processo de confirmação de transação e do bloco, garantindo a segurabilidade de todo o sistema.

3.2.4. Comparativo

Abordadas as três plataformas apresentadas neste minicurso: Ethereum, Hyperledger e MultiChain, torna-se relevante a realização de um comparativo que contenha as principais e mais atrativas configurações de cada uma dessas plataformas. Como características para comparação, apresenta-se na Tabela 3.4 as características:

- Modelos Blockchain: com base nas sua características, classificar o Blockchain nos modelos conhecidos.
- Versão Blockchain: As versões Blockchain que são aplicadas na plataforma.
- Linguagem de programação: Quais as principais linguagens de programação que realizam interações com a plataforma.
- Mecanismos de Consenso: Os mecanismos de consenso que são aplicados e utilizados pela plataforma.
- Licença: Se a aplicação possui licença aberta ou proprietária.

Tabela 3.4: Características das plataformas que utilizam a tecnologia Blockchain.

Plataformas	Modelo	Versão	Linguagem	Consenso	Licença
Ethereum	Pública/Privada	Blockchain 1.0/2.0/3.0	C++, Go, JavaScript,Python	PoW e PoS	Aberto
Hyperledger	Privada	Blockchain 3.0	C++, Java, Python, Ruby	PBFT e outros[2]	Aberto
Multichain	Privada	Blockchain 2.0/3.0	C#, JavaScript, PHP, Python, Ruby	PBFT	Aberto

A partir da Tabela 3.4 é possível observar que a plataforma Hyperledger possui o maior escopo quando se relaciona a aplicações descentralizadas. No entanto, possui maior exigência de configuração e adequação para o seu próprio problema. Já as plataformas Ethereum e MultiChain possuem seu principal escopo voltado para contratos inteligentes a partir da aplicação de transações, tornando-as mais adequadas para agilidade no desenvolvimento de aplicações e mais simples de serem configuradas. Contudo, torna-se necessário o conhecimento das vulnerabilidades que atingem a tecnologia Blockchain, pois estas podem influenciar o bom funcionamento destas redes desenvolvidas com o uso da tecnologia.

3.2.5. Ataques e vulnerabilidades envolvidos com Blockchain

Com a necessidade de investigar as principais vulnerabilidades conhecidas na tecnologia, é realizada uma revisão bibliográfica que identifica os principais ataques e vulnerabilidades em relação ao uso de Blockchain. A Tabela 3.5 é dividida pelos seguintes campos: Nome do ataque/vulnerabilidade, versão Blockchain, categoria e relação com a segurança.

[2] A plataforma Hyperledger permite o desenvolvimento e aplicação, implementados, de vários consensos a escolha do desenvolvedor.

Tabela 3.5: Principais ataques e vulnerabilidades identificadas relacionadas a Blockchain classificadas em categorias.

Ataque/Vulnerabilidade	**Versão Blockchain**	**Categoria**	**Relação**
"Vulnerabilidade 51%"	Blockchain 1.0, 2.0, 3.0	Redes	Imutabilidade/Procedência
Chave Privada de Segurança	Blockchain 1.0	Usuário	Cifragem
DDoS	Blockchain 1.0, 2.0, 3.0	Redes	Transações/Procedência
Gasto Duplo	Blockchain 1.0, 2.0, 3.0	Redes	Arquitetura
Ataque Eclipse	Blockchain 1.0, 2.0, 3.0	Redes	Arquitetura/Transação/Procedência
Ataque de Vivacidade (*Liveness*)	Blockchain 1.0	Poder Computacional	Transações/Procedência
Mineração Egoísta	Blockchain 1.0, 2.0, 3.0	Poder Computacional	Procedência
Otimização *Smart Contract*	Blockchain 1.0, 2.0, 3.0	Usuário	Transparência/Transação
Privacidade de Transação	Blockchain 1.0	Redes	Arquitetura
Retenção de Blocos	Blockchain 1.0, 2.0, 3.0	Poder Computacional	Transparência
Smart Contract Malicioso	Blockchain 2.0	Usuário	Procedência/Transações

A partir da Tabela 3.5 é possível identificar as principais vulnerabilidades conhecidas na tecnologia Blockchain e quais versões da tecnologia estão sobre influencia destas vulnerabilidades. Analisando as vulnerabilidades identificadas (Tabela 3.5), foi realizada uma classificação inicial associando estas três categorias principais: Redes, Poder Computacional e Usuário.

1. **Redes:**
   Aspectos relacionados ao controle dos nós, forma das transações ou até mesmo incapacitação operacional da rede.

   (a) DDoS: O objetivo do atacante é tornar o serviço indisponível durante o processo de ataque. Os sistemas de defesa contra (DDoS) normalmente não são capazes de resistir sozinhos contra ataques em larga escala [37]. É importante notar que devido a natureza totalmente distribuída/replicada de Blockchains, estes se tornam naturalmente resilientes a ataques de negação de serviço distribuídos. Além disto, o modelo de custo associado a Blockchains públicos (impondo taxas para as transações), evita que usuários maliciosos realizem o envio em massa de transações impondo um alto custo a este tipo de ataque. *Distributed Ledger*s (DLTs) que não possuem um modelo de custo associado a cada transação estão suscetíveis a ataques de negação de serviço por parte de nos maliciosos dentro de um consorciou ou organização. Porém, entidades com permissões de escrita em uma DLT possuem um certo grau de confiança dentro da organização/consórcio.

   (b) Ataque Eclipse: O Ataque Eclipse tem como intenção ganhar controle sobre os nós, desta forma controlando grande maioria do tráfego da rede. Quando um ataque Eclipse é bem sucedido, permite que o invasor controle todo tráfego de sobreposição, permitindo negação arbitrária de serviço ou de censura [38].

   (c) Gasto Duplo: Esta vulnerabilidade está diretamente atribuída às criptomoedas, nas quais atacantes fazem múltiplas transações com a mesma moeda. Para este ataque ser realizado é necessário que o atacante minere privativamente, tentando estender ao máximo o bloco minerado sem publicar o cálculo, então é transmitida a transação para a organização de interesse e esperar para que a

transação seja registrada, e então minerado até que este bloco seja maior do que o bloco público, assim é publicado o cálculo apagando a transação feita com a organização [39]. O problema do gasto duplo ocorre de maneira diferente para os diferentes tipos de mecanismos de consenso. Por exemplo, em um Blockchain baseada em PoW, o gasto duplo exige que o atacante tenha controle sobre 51% do poder computacional da rede, decidindo, deste modo, priorizar uma cadeia de blocos em detrimento a outra. No entanto, em um Blockchain baseada em em PoS, na qual não se exige um gasto de recursos como prova de trabalho, a rede fica mais suscetível a criação de cadeias paralelas (uma vez que estas cadeias podem ser geradas sem esforço computacional). Neste caso, a alternativa para Blockchains baseados em PoS é um mecanismo de consenso híbrido entre PoS e PoW no qual introduz-se um esforço computacional para a geração de blocos, mas mantendo a determinação dos mineradores baseando-se em seus recursos ("*stake*").

(d) Privacidade de Transação: Esta vulnerabilidade está relacionada com a possibilidade de rastreabilidade do Blockchain, conforme o modo de programação o destinatário pode ser detectado através da transação da rede [40]. A maioria das criptomoedas públicas utilizam um esquema de pseudo-anonimidade, na qual usuários são identificados por endereços (*hash* gerado criptograficamente) não revelando, deste modo, sua identidade. Neste sentido, é possível observar todas as transações financeiras de contas mas não a identidade do usuário que gerencia a conta. No entanto, a conversão de uma criptomoeda para um dinheiro final requer a utilização de uma casa de cambio, que por sua vez exige a identificação do usuário, e assim estabelecer uma relação entre usuário-conta. No entanto, existem criptomoedas como Monero [41] que utiliza um protocolo (CryptoNote) que obfusca por meio de uma primitiva baseada em *ring signatures* as três partes essenciais de uma transação: remetente, valor e destinatário.

(e) "Vulnerabilidade 51%": Este ataque é realizado a partir do mecanismo de consenso do Blockchain, em que se o invasor obter 51% do *hashing* do *pool* ele tem o controle sobre o bloco. A partir disto, existe a possibilidade de realizar mudanças entre outras questões [40].

2. **Poder Computacional:**
Ataques relacionados ao aumento de *hashing* tem o intuito de obter benefícios sobre mineradores honestos ou simplesmente reduzir a recompensa que um grupo de mineradores tem direito.

(a) Retenção dos Blocos: O objetivo deste ataque é sabotar mineradores honestos, fazendo com que desistam do *pool*. O minerador inicia o processo como um minerador honesto, mas o atacante envia apenas uma parcial da prova de trabalho. Se encontrar uma solução completa que constitua uma prova de trabalho descarta a solução, reduzindo o rendimento total [42].

(b) Mineração Egoísta: Este ataque tem como finalidade obter recompensa ou perda de poder computacional de mineradores honestos. Especificamente, a

mineração egoísta força os mineradores honestos a gastar seus ciclos computacionais em blocos destinados a não fazer parte do Blockchain. Mineradores egoístas atingem esse objetivo revelando seletivamente seus blocos minados para invalidar o trabalho dos mineradores honestos [43].

(c) Ataque de Vivacidade(*Liveness*): Este ataque é realizado para atrasar o máximo possível o tempo de confirmação de um alvo transação [44]. Este ataque passa por três fases: a primeira que é quando o minerador malicioso cria vantagem sobre os mineradores honestos, a segunda é de negação de serviço (*Denial-of-Service* (DoS)) e a terceira que é a retardadora do Blockchain.

3. **Usuário**:
   Ataques relacionados a categoria usuários são relacionados a programação, intenções maliciosas e até mesmo falta de conhecimento.

   (a) Chave Privada de Segurança: Este ataque é relacionado a chave de segurança privada de cada usuário. Caso a chave for roubada ou perdida o usuário dificilmente conseguirá recupera-lá.

   (b) *Smart Contract* Criminoso: Este contrato pode facilitar o vazamento de informações confidenciais, roubo de chaves criptográficas e vários tipos de comportamentos do usuário [40].

   (c) Otimização *Smart Contract*: Relacionado a programação do contrato, em que este é pouco otimizado causando grandes perdas para quem o utiliza. Segundo [45] foram detectados alguns padrões em códigos que demonstram funções inúteis, que não são utilizadas, e também códigos em *loop*.

A classificação apresentada foi dividida em apenas três categorias, que de modo geral, abrangem as principais necessidades que são conhecidas pelo Blockchain. Há autores que realizam a classificação em cinco diferentes categorias, explorando um maior número de vulnerabilidades, mas vale ressaltar que em uma classificação completa estas vulnerabilidades podem ser divididas em sub-categorias para maior refinamento das vulnerabilidades.

3.3. Analisando mecanismos de consenso PoW e PBFT quanto a DoS

A partir dos dados apresentados, nas Seções 3.1 e 3.2, é perceptível que o Blockchain possui uma adaptabilidade e versatilidade para várias realidades e necessidades diferentes. Porém, a tecnologia apresenta algumas adversidades relacionadas as questões de custo computacional e também às questões de segurança, que variam de acordo com o modelo de aplicação.

Com essa premissa, torna-se necessário o estudo de viabilidade na aplicação do Blockchain, as necessidades da instituição, assim como as tecnologias aplicadas em conjunto com o Blockchain. Quanto ao Blockchain é necessário considerar as questões relacionadas a tecnologia como: Modelo, versão, mecanismos de consenso, etc.

3.3.1. Definição e motivação

O crescimento exponencial da tecnologia Blockchain é incontestável, as demandas por aplicações da tecnologia possuem diferentes contextos e realidades de usuários. Quanto a estas aplicações do Blockchain, busca-se melhorar a qualidade e eficiência da tecnologia, mas também existe a preocupação ligada ao custo computacional que é necessário para o funcionamento da tecnologia Blockchain. Neste sentido, percebe-se que há diversos mecanismos de consenso, que são responsáveis em realizar várias operações e entre estas está o processo de validação do Blockchain. Contudo, não foram encontrados estudos que tenham como objetivo analisar diferentes mecanismos de consenso do Blockchain em contexto com nenhuma outra tecnologia.

Como exemplo de aplicação, possui-se dois mecanismos de consenso que possuem seus benefícios e problemas, como o PBFT e o tradicional mecanismo PoW. Para avaliar o impacto que a escolha do mecanismo possui, há a necessidade de entender sobre os aspectos da tecnologia utilizada. Isto é, se é necessário aplicar o Blockchain ou não, os usuários envolvidos e também que atenda as demandas e objetivos das aplicações. Exemplificando, aplicar um Blockchain para registrar operações de alocação de VM em nuvens computacionais implica em obter requisitos de transparência, procedência dos dados, proteção e gerenciamento, mudança de modelo de ameaça, *etc* [46].

A decisão de da escolha de qual tecnologia aplicar é complexa, impactando diretamente no desempenho e possíveis vulnerabilidades da aplicação. Além das ameaças e vulnerabilidades padrões que diversas tecnologias possuem, percebe-se, a partir, da Tabela 3.5 alguns dos ataques e vulnerabilidades existentes para a tecnologia Blockchain. Com este levantamento, torna-se relevante analisar os mecanismos de consenso Blockchain junto aos aspectos de segurança definidos por organizações, como o *National Institute of Standards and Technology* (NIST).

No atual cenário do uso da tecnologia Blockchain, há incertezas quanto a viabilidade da aplicação da tecnologia, principalmente pelo fato de sua popularidade, de qual melhor modelo e mecanismo deve-se aplicar com outras tecnologias, principalmente pela variabilidade de possibilidades existentes. O problema em questão é a falta de critérios para comparar os mecanismos de consenso do Blockchain e sua relação com vulnerabilidades, ameaças e riscos existentes que são intrínsecos a tecnologias e arquiteturas empregadas na solução Blockchain.

3.3.2. Motivação da análise

A tecnologia Blockchain tem apresentado considerável versatilidade no contexto de possíveis aplicações, tornando-se benéfica para diferentes contextos, *e.g.*, contratos inteligentes, internet das coisas (IoT), segurança da informação, *etc* [1, 2]. A utilização do Blockchain de forma paralela a estas tecnologias tem proporcionado um ambiente íntegro, seguro, descentralizado e transparente [47].

A proposta de experimento neste trabalho consiste na realização de uma breve análise de segurança dos mecanismos de consenso PoW e PBFT aplicados ao Blockchain quando submetidos ataques simples de DoS. Dentre as necessidades atuais das tecnologias baseadas em Blockchain estão aspectos como: auditoria, procedência dos dados,

proteção e gerenciamento dos dados e a segurança no gerenciamento do ciclo de vida das informações. Em que estes aspectos relevantes podem ser supridos com o emprego do Blockchain, que disponibiliza meios para incorporar questões como transparência, autonomia, imutabilidade, anonimado, descentralização dentre outros.

Atualmente, há pesquisas que indicam que os principais responsáveis pelas violações de dados que ocorrem em nuvens computacionais são causadas por pessoal internos da instituição que está conectado diretamente a sua rede local [48, 49]. Este fato justifica uma necessidade de análises voltadas para soluções Blockchain baseadas no Modelo Privado e de Consórcio. Outro preceito para a realização das análises são as vulnerabilidades apresentadas na Seção 3.2.5, que delimitam em grupos a realização da análise.

A escolha do ataque a ser aplicado pertence ao grupo de redes, que envolvem ataques como: DoS, Ataque Eclipse, Gasto Duplo, Privacidade das Transações e a Vulnerabilidade "51%". As questões relacionadas a este grupo impactam no desempenho, funcionalidade, custo computacional e podem atingir não somente a rede Blockchain, mas todo o sistema em si [50].

O intuito desta análise é a identificação das melhores alternativas de aplicação para os mecanismos de acordo com o contexto do Blockchain. Com o objetivo de facilitar que usuários da tecnologia realizem integrações ao seus sistema com uma tecnologia segura, eficiente, viável, custo benefício, desempenho e funcionalidade para o seu cenário.

3.3.3. Critérios de Análise

A partir das atuais necessidades das tecnologias que podem ser aplicadas em conjunto com o Blockchain, nos trabalhos relacionados e nos ataques e vulnerabilidade listados na Subseção 3.2.5, foi possível identificar e analisar as principais preocupações com os mecanismos de consenso para segurança do Blockchain. Com base nestas preocupações, foram definidos os critérios:

- Procedência dos dados: A procedência dos dados é um processo de auditoria que mantém um registro, não somente de *logs*, mas de todas operações realizadas com um objeto. Basicamente a partir da procedência é possível compreender tudo que foi realizado com determinado objeto e suas possíveis alterações e até mesmo fraudes. De acordo com o *Cloud Security Alliance* (CSA) os grandes desafios das nuvens computacionais são: Visibilidade detalhada, maior escopo de aplicação e transparência reduzida. A garantia de procedência dos dados tem como visão auxiliar que estes desafios possam ser minimizados, para então auxiliar o provedor da rede e também ao usuário possíveis garantias de que seu conteúdo esteja protegido e somente com as reais modificações que foram aprovadas pelos mesmos.

  Quanto a verificabilidade da confiabilidade do sistema, baseia-se pela inviolabilidade do mecanismo de consenso. A segurança do algoritmo de consenso é baseado no problema dos generais bizantinos [51], em que o Blockchain utiliza um livro-razão no qual todas alterações realizadas em determinado objeto é feita a conferência das assinaturas digitais para geração das transações, após isso a atualização e validação dos blocos. Alguns ataques descritos na Subseção 3.2.5 podem gerar problemas de funcionalidade ao consenso, ataques como: "Vulnerabilidade 51%",

Ataque Eclipse, DDoS, Ataque de Vivacidade, *etc*. Ataques estes que levam a necessidade de verificação dos mecanismos de consenso como seguros para determinadas aplicações.

A verificação desta inviolabilidade é feita a partir do livro-razão e da verificação do tamanho de bytes dos objetos. Para garantir que realmente nenhuma mudança realizada através dos ataques seja validada e represente periculosidade ao sistema.

- Controle e Gerenciamento do Nó: O controle de nó é um processo que visa obter o controle sob todo o nó, permitindo a existência de vulnerabilidade e de alterações sobre o bloco. Uma parte considerável dos problemas voltados as vulnerabilidades da tecnologia Blockchain vistos na Subseção 3.2.5 são causados por problemas de segurança que podem ser evitados com maior eficiência e controle dos nós, evitando desta forma que conteúdos protegidos possam ser aceitos, acessados ou alterados. Alguns ataques como: Ataque Eclipse, DDoS e Mineração Egoísta podem dificultar a validação de consenso ou até mesmo gerar alterações nas transações para determinados objetos.

  Quanto a forma de medidas para determinada questão, são utilizados ataques que promovam controle do nó ou que instabilizem a rede para após o término que possa ser verificado a influência do ataque de modo geral na rede Blockchain. A principal forma de identificação da eficiência do ataque é obtida a partir de dados coletados durante a execução do mesmo e também a partir de transações e blocos que sejam validados ou contestados perante a rede.

3.4. Ambiente de Experimentação

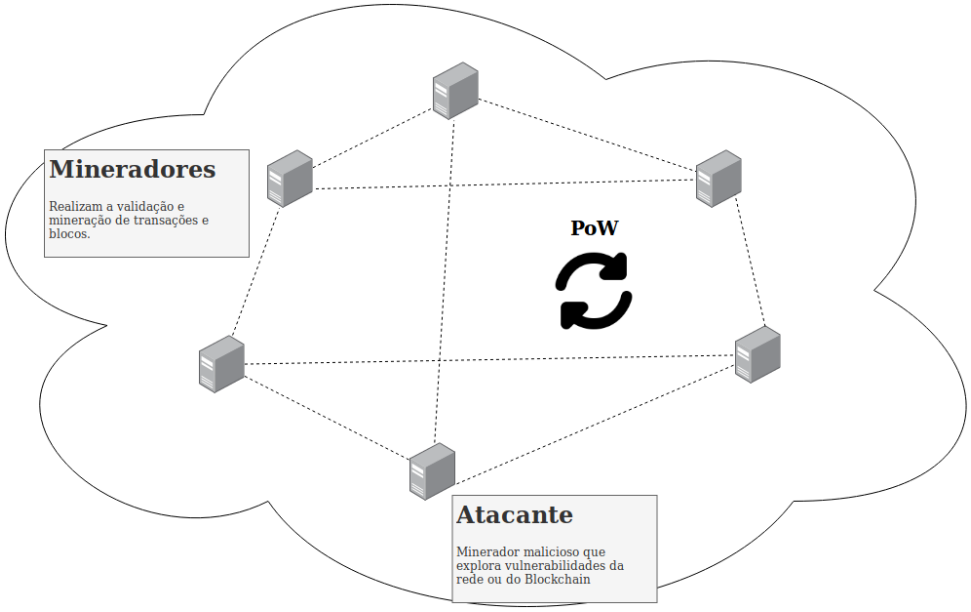
Os experimentos visam mostrar como alguns ataques podem explorar algumas soluções Blockchain e como mecanismos de consenso são afetados. Deste modo, foi definido um plano de testes e um ambiente de testes que possibilitasse a execução e replicação dos experimentos de modo facilitado. Todos os cenários usam como base uma topologia *flat* com seis VMs, todas com a distribuição GNU/Linux Ubuntu Server 16.04 LTS.

3.4.1. Plano de Testes

Este plano de testes foi desenvolvido baseando-se nos critérios estabelecidos na Seção 3.3. O objetivo é realizar análise dos métodos de consenso Blockchain, para isso são empregados dois cenários, as configurações de ambiente de testes utilizadas são as exigidas pelos sistemas Blockchain Ethereum e Multichain, a distinção destes cenários é, essencialmente para estes experimentos, o mecanismo de consenso aplicado. Os cenários definidos são:

- Cenário 1 - Ethereum: O cenário utiliza o modelo de rede privada de Blockchain, em que as seis instâncias realizam o processo de mineração e validação de transações do Blockchain (Figura 3.6).

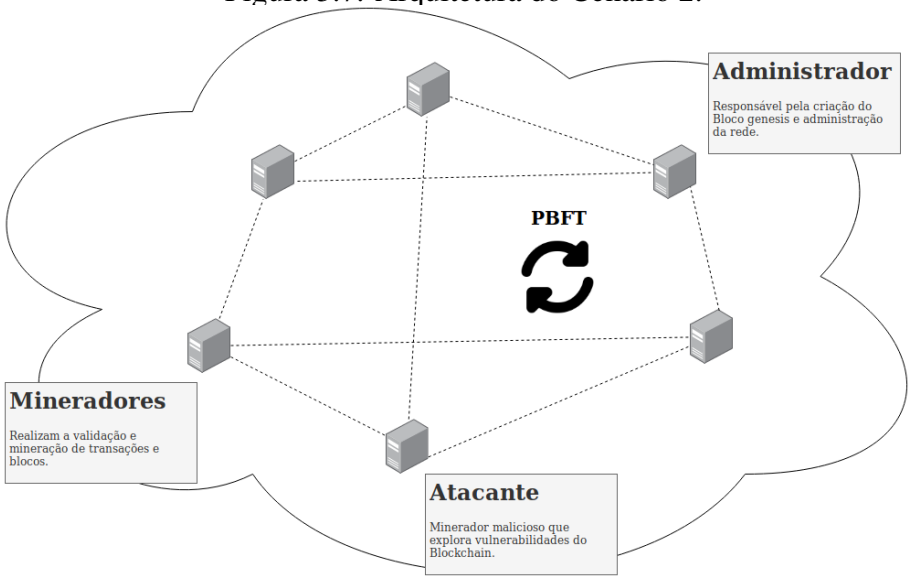
Figura 3.6: Arquitetura do Cenário 1.



O objetivo desta experimentação é investigar as características e comportamentos iniciais do mecanismo de consenso PoW. A partir da análise inicial de comportamento, uma destas instâncias se tornara atacante tendo em vista as vulnerabilidades da rede e da implementação do Blockchain. A partir da finalização do experimento, é realizado um processo de análise do cenário após a execução dos ataques. O intuito deste experimento é averiguar a procedência dos dados, controle e gerenciamento dos nós e por fim a estabilidade da rede, identificando desta forma as influências do ataque em relação com o funcionamento da rede.

- Cenário 2 - Multichain: O cenário possibilita que as seis instâncias realizem o processo de mineração e validação de transações e blocos, mas somente uma destas VM possui o nível de administrador, o que a torna parcialmente descentralizada (Figura 3.7).

Figura 3.7: Arquitetura do Cenário 2.



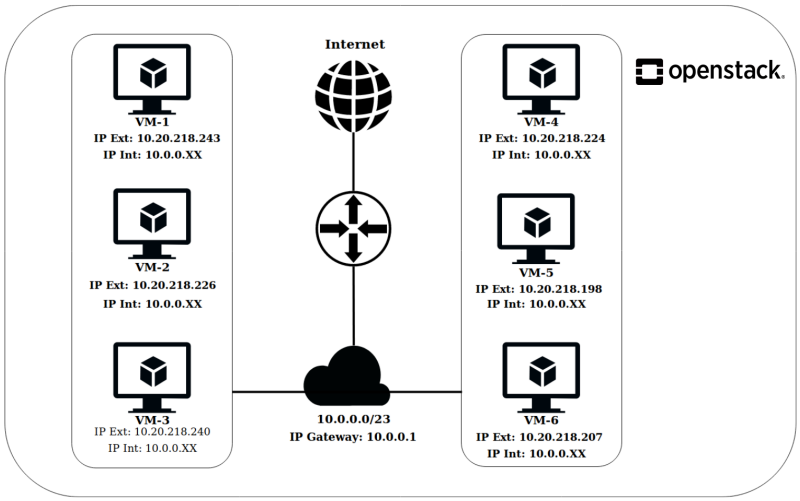
O Cenário 2 possui o objetivo de investigar as características e comportamentos da rede Blockchain privada e parcialmente descentralizada com o mecanismo de consenso PBFT. Ao fim das análises iniciais do comportamento da rede, uma destas instâncias se torna atacante que tem como objetivo comprometer o funcionamento da rede e do Blockchain. Após a realização do ataque, um processo de análise do cenário é realizado a partir dos dados coletados durante a execução destes. O experimento tem como objetivo de averiguar a procedência dos dados, o controle/gerenciamento dos nós e também a estabilidade da rede. O intuito deste experimento é a identificação das influências causadas pelo ataque no sistema, em relação ao seu funcionamento e consenso.

Quanto as VMs em ambos ambientes não possuem habilitados recursos de memória virtual *swap*, utilizando apenas a memória principal. A escolha desta abordagem é necessária pois para obtenção de informações mais claras e com maior facilidade é necessário o esgotamento dos recursos.

3.4.2. Arquitetura do Ambiente

O Laboratório de Processamento Paralelo e Distribuído (LabP2D) possui uma nuvem computacional *Infrastructure as a Service* (IaaS) baseada na solução aberta OpenStack versão Mitaka, com nome de Nuvem Tchê. Quanto ao ambiente de testes, o mesmo foi construído como um projeto OpenStack na Nuvem Tchê, tal que a abordagem escolhida foi o uso executado de VMs para execução das soluções Ethereum e Multichain usando o Modelo Privado. Embora não seja comum o uso de um sistema Blockchain inteiro em uma nuvem apenas, a escolha se deu pela praticidade de execução dos experimentos e isolamento de outros fatores (e.g., tráfego de *background*) que podem tornar a análise mais subjetiva e complexa. Pensando nisso, a arquitetura foi construída através de três componentes principais: Roteador, Rede e VM de acordo com a Figura 3.8. Quanto a arquitetura da rede utilizada, é uma rede *overlay*, que é conectadas por enlaces e *switches* virtuais no OpenStack. É importante ressaltar que não foram criados mais roteadores (saltos) ou inserida latência, pois os experimentos visam averiguar o consumo de recursos.

Figura 3.8: Arquitetura de ambiente de testes no OpenStack.



A partir da Figura 3.8 é possível observar que a comunicação com a Internet é realizada através do Roteador que possui como interface interna o IP 10.0.0.1 que é do *gateway* da Rede interna 10.0.0.0/24. Na rede interna possui as instâncias executando o GNU/Linux Ubuntu Server 16.04 LTS, que possui diferentes IP 10.0.0.3-20 denominados entre as VM-1 a VM-6. Sobre estas VMs foram instaladas diferentes plataformas de Blockchain, constituindo um Blockchain usando o Modelo Privado. Vale mencionar que, dependendo do tipo de ataque realizado, é necessário que mais de uma VM possa realizar os ataques que são mencionados na Seção 3.3.

3.5. Experimentos aplicados

Para cada um dos cenários é apresentado o objetivo do experimento, fluxo da rede, comportamento do mecanismo de consenso, os ataques realizados e por fim os possíveis problemas que foram causados na plataforma.

3.5.1. Cenário 1

O primeiro cenário de experimento é utilizado a plataforma Ethereum com seis VMs que utilizam sistema operacional GNU/Linux Ubuntu Server 16.02 LTS. O Blockchain Ethereum é um sistema que por padrão aplica o modelo público do Blockchain, mas permite que o sistema utilize os modelos privado e de consórcio.

A tecnologia utiliza uma rede totalmente descentralizada, na qual diferentes pares se conectam e realizam trocas de informações entre si, estas trocas de informações utilizam os protocolos de rede TCP e UDP. Na Listagem 3.1 é apresentado o formato de um bloco usado neste cenário.

Listagem 3.1: Informações do Bloco

```
eth.getBlock(6821)
{
 difficulty: 2871846,
 extraData: "0xd88301081b846765746888676f312e31302e34856c696e7578",
 gasLimit: 8000000,
 gasUsed: 21000,
 hash: "0x85eef246d6b1fb07e415160bb0cd965c45e58025dcd82b6f26a66adf7905aaab",
 logsBloom: "0x00000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000000000000000000000000
     0000",
 miner: "0xea1cefe73fd12583bfd7911db2d4726a3aedee0b",
 mixHash: "0xa0a129927080a4f81dfe1033b5e7a3c087ec992a315e79a563124425f3b48de0",
 nonce: "0x699c8a2dbccdf163",
 number: 6821,
 parentHash: "0x5928b23f71c07af600236c52978f95cb438a8853fcd2a41ca0b6d2359f2fa4f
     7",
 receiptsRoot: "0x13cbb7efc83a39275d711a65415daa8d9050f62143a257a82be6a10e0c77
     da7f",
 sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d4934
     7",
 size: 655,
 stateRoot: "0xb05a6fc64f193b45d4bf068d3635f8c00dbfd9c7830f8e7feb40feebe8b1f5c8
     ",
 timestamp: 1559889720,
 totalDifficulty: 6566222689,
 transactions: ["0xfe9e7da787548c7160cd34bc0bac0450a62a29031a8647398a613e1c1809
```



```
        bafa"],
21    transactionsRoot: "0x17e7db8c04457551df93e6e9fee5a286aad50e6b93d28e1b6303628
        aed12248c",
22    uncles: []
23  }
```

A partir da Listagem 3.1 observa-se informações disponíveis para consulta sobre os blocos que pertencem a rede Blockchain. Há informações relevantes, como a linha 9, em que apresenta a carteira utilizada para a mineração do bloco, já as linhas 20 e 21 apresentam as transações que foram realizadas e estão registradas e auditadas. Quanto a realização de uma transação, é possível identificá-la na Listagem 3.2.

Listagem 3.2: Informações de uma transação

```
1   eth.getTransaction("0xfe9e7da787548c7160cd34bc0bac0450a62a29031a8647398a613e1c1
        809bafa")
2   {
3     blockHash: "0x85eef246d6b1fb07e415160bb0cd965c45e58025dcd82b6f26a66adf7905aaab
        ",
4     blockNumber: 6821,
5     from: "0xea1cefe73fd12583bfd7911db2d4726a3aedee0b",
6     gas: 90000,
7     gasPrice: 1000000000,
8     hash: "0xfe9e7da787548c7160cd34bc0bac0450a62a29031a8647398a613e1c1809bafa",
9     input: "0x",
10    nonce: 0,
11    r: "0xc56d5e83f9d21a2e6a36c881194c2fbc41e847b41f016c21698d4349e786b525",
12    s: "0x16086477c6640e9d922b41ae9b2ec3f3118c118769f0af75ec792201631d8e83",
13    to: "0xf6a285239af6faa74e1686fac7c317192cd88768",
14    transactionIndex: 0,
15    v: "0x75bcd186",
16    value: 10000000000000000000
17  }
```

Na Listagem 3.2 é verificado uma transação realizada entre a VM-1 a partir da carteira, presente na linha 5, para a VM-2 em que utiliza a carteira apresentada na linha 13. As informações que são fornecidas para consulta, permitem a verificação e validação não somente da transação, mas também do bloco em que esta encontra-se indexada.

A partir das Listagens 3.1 e 3.2 é possível averiguar o estado inicial da arquitetura e do comportamento desta rede Blockchain privada. Seis VMs participam do processo de validação das transações e dos blocos com o mecanismo de consenso PoW, possibilitando o monitoramento e auditoria de toda movimentação e mudanças ocorridas nos objetos.

Estas informações permitem a verificação da existência e validação não somente da transação, mas também do bloco que ela pertence com associação as carteiras que foram envolvidas durante o processo. Ressalta-se que estas carteiras apresentam somente os dados das mesmas, garantindo anonimato total dos dados dos envolvidos.

3.5.1.1. Ataque de Negação de Serviço

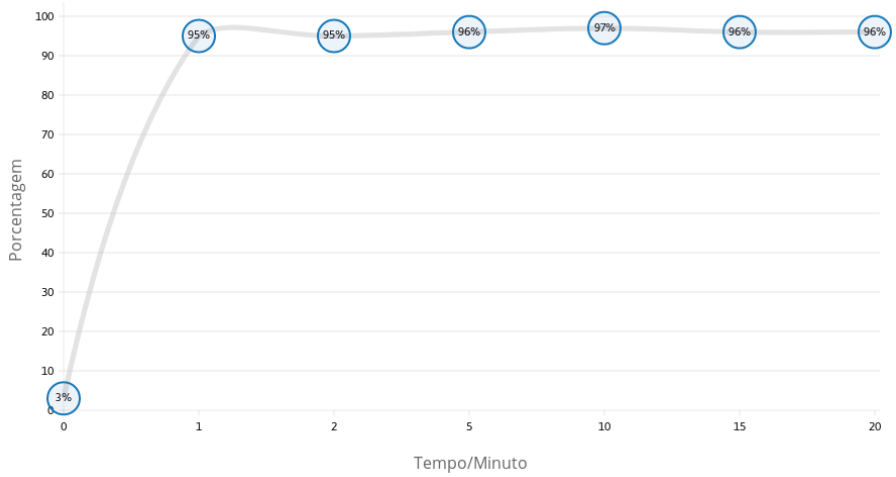
Foram escolhidos dois ataques:

- *Transactions Flood*: realizado a partir da VM-2 e VM-6 endereçados para a carteira presente na VM-1. O Blockchain Ethereum, como característica sua, não possui um administrador da rede ou um nó centralizado que monopoliza e distribui os

processos para criação e validação de blocos e transações. Com a falta de um meio centralizados, o ataque em um Blockchain usando o Modelo Privado não afetou em proporções consideráveis o desempenho de memória e processamento da VM, pois a descentralização permite que a rede funcione normalmente, validando as transações e minerando novos blocos.

- *UDP Flood* foi realizado a partir da VM-6 que atacavam a VM-1. O Ethereum utiliza o serviço *User Datagram Protocol* (UDP) para troca de informações entre os nós. Uma aspecto relevante quanto ao uso do mecanismo PoW, em um Modelo Privado, é a não necessidade do uso de uma GPU, deixando a CPU responsável pela mineração dos blocos, que é observado a partir da Figura 3.9.

Figura 3.9: Análise do consumo de processador, sem o ataque *UDP Flood*.



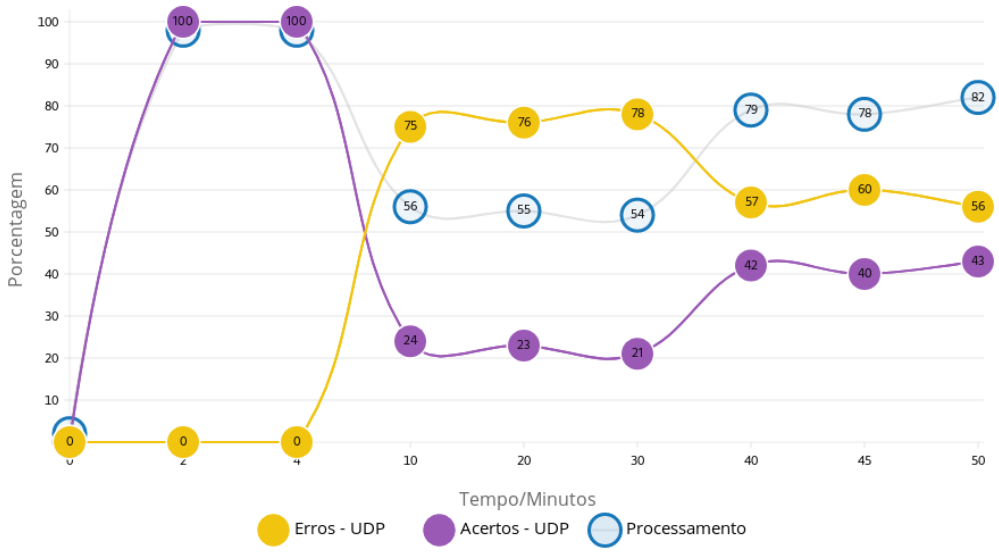
Os dados apresentados na Tabela 3.6 e pela Figura 3.9 é possível observar, a partir dos pontos, o consumo do processamento da VM, que é realizado pela mineração de novos blocos, vale lembrar que a Figura 3.9 não há a ocorrência de nenhum ataque, somente o consumo normal do processamento.

Tabela 3.6: Tabela do consumo de processador sem o ataque *UDP Flood*.

Tempo/Min	0	1	2	5	10	15	20
Processamento	3,1%	95,74%	95,08%	96,47%	97,40%	96,56%	96,89%

A Figura 3.10 apresenta o consumo da *Central processing unit* (CPU) quando a instância VM-1 sofre um ataque *UDP Flood*.

Figura 3.10: Análise do consumo do processador durante o ataque *UDP Flood*.



A partir da Figura 3.10 e Tabela 3.7 observa-se três pontos: O consumo da CPU do sistema, Pacotes UDP que foram recebidos/respondidos e o terceiro indica os pacotes UDP que apresentaram erro. Quanto ao consumo de processamento, é notável que seu percentual de uso cai consideravelmente com o aumento da quantidade de pacotes UDP com erros, é observado no sistema que a partir desta diminuição do uso de processamento paralelamente ocorre uma diminuição significativa na mineração e validação de blocos e transações ou até mesmo a mineração é suspensa temporariamente.

Tabela 3.7: Tabela do consumo de processador durante o ataque *UDP Flood*.

Tempo/Min	0	2	4	10	20	30	40	45	50
Processamento	2,3%	98,78%	98,70%	56,28%	55,86%	54,56%	79,73%	78,39%	82,45%
Acertos - UDP	0%	100%	100%	24,5%	23,7%	21,7%	42,3%	40,0%	43%
Erros - UDP	0%	0%	0%	75,5%	76,3%	78,3%	57,7%	60,0%	56,8%

Na Figura 3.11 e Tabela 3.8 é possível observar um crescimento do nível de consumo de memória, que é gerado a partir das filas de requisição da troca de informações entre os nós da rede Blockchain. Contudo, a mesma apresenta estabilidade no consumo durante a realização do ataque, indicando que o consumo de memória não é afetado durante o processo.

Figura 3.11: Análise do consumo de memória durante o ataque *UDP Flood*.

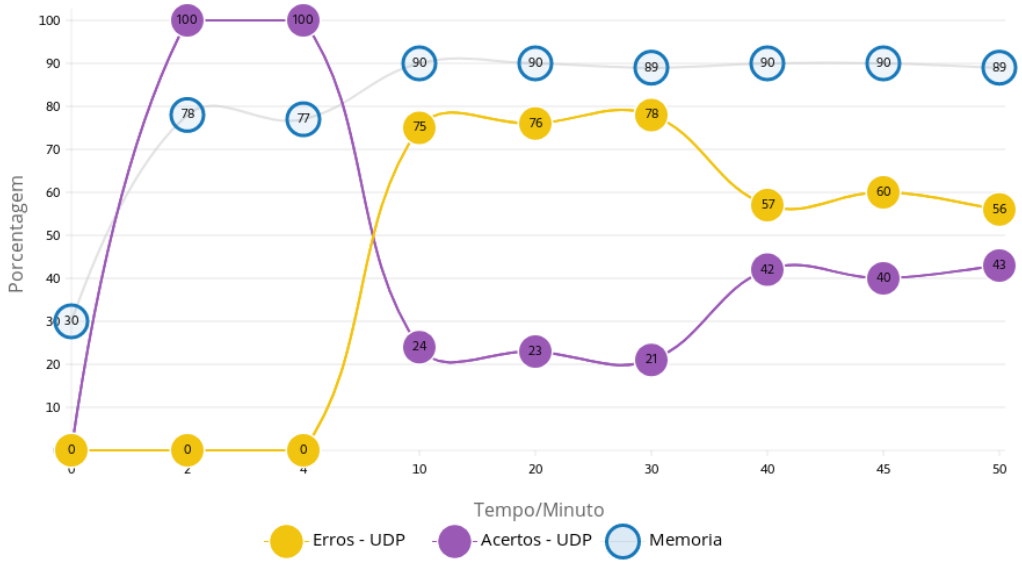


Tabela 3.8: Tabela do consumo de memória durante o ataque *UDP Flood*.

Tempo/Min	0	2	4	10	20	30	40	45	50
Memoria	30,00%	78,7%	77,30%	90,84%	90,30%	89,45%	90,14%	90,37%	89,92%
Acertos - UDP	0%	100%	100%	24,5%	23,7%	21,7%	42,3%	40,0%	43,0%
Erros - UDP	0%	0%	0%	75,5%	76,3%	78,3%	57,7%	60,0%	56,8%

A Figura 3.12 apresenta o tráfego *Transmission Control Protocol* (TCP) durante o período do ataque *UDP Flood*. Observa-se nos dados da Tabela 3.9 que ocorre um fluxo mais intenso na rede TCP, identificando desta forma possíveis atrasos na troca de informações e pacotes entre os nós da rede Blockchain.

Figura 3.12: Análise de tráfego TCP durante o ataque *UDP Flood*

Tabela 3.9: Análise de tráfego TCP durante o ataque *UDP Flood*.

Tempo/Min	0	2	4	10	20	30	40	45	50
Entrada/seg	0,0	15,77	11,60	1195,7	1361,83	919	128	12,13	9,0
Saida/seg	0,0	19,57	14,5	4691,03	5241,37	3132,47	9,83	13,37	8,10

Este experimento permite revelar algumas das implicações do ataque em relação ao desempenho e operação do Blockchain Ethereum. Com um esforço limitado (uso de poucos nós equivalentes) por parte do atacante, é possível detectar uma redução considerável na taxa de mineração e também impacto na troca de informações da VM-1, atacada, com os outros nós pertencentes a rede.

3.5.2. Cenário 2

Este cenário de experimento utiliza a plataforma Multichain com a versão 1.0.5 e possui seis VMs que utilizam sistemas operacionais GNU/Linux Ubuntu Server 16.02 LTS. O Blockchain Multichain é um sistema projetado para sua aplicação com o Modelo Privado e Consórcio. O Multichain utiliza uma rede parcialmente descentralizada, que possui a existência de um administrador criador do Bloco Gênesis, em que este possui o poder de permitir diferentes conexões na rede Blockchain e que também permite o recebimento, envio e mineração de blocos na rede. As VMs aplicadas no ambiente utilizam o protocolo de rede TCP para realizarem trocas de informações entre si. A Listagem 3.3 apresenta as principais informações sobre a rede Blockchain Multichain.

Listagem 3.3: Informações da Rede

```
1  {"method":"getinfo","params":[],"id":1,"chain\_name":"chain1"}
2  {
3  "version" : "1.0.5",
4      "nodeversion" : 10005901,
5      "protocoloversion" : 10011,
6      "chainname" : "chain1",
7      "description" : "MultiChain chain1",
8      "protocol" :    "multichain",
9      "port" : 7317,
10     "setupblocks": 60,
11     "nodeaddress" : "chain1@10.0.0.3:7317",
12     "burnaddress" : "1XXXXXXXGiXXXXXXTNXXXXXX64fp8",
13     "incomingpaused" : false,
14     "mininpaused" : false,
15     "walletversion" : 60000,
16     "balance" : 0.00000000,
17     "walletdbversion" : 2,
18     "reindex" : false,
19     "blocks" : 507,
20     "timeoffset" : 0,
21     "connections" : 6,
22     "proxy" : "",
23     "difficulty" : 0.00000006,
24     "testnet" : false,
25     "keypoololdest" : 1557377147,
26     "keypoolsize" : 2,
27     "patyxfee" : 0.0000000,
28     "relayfee" : 0.0000000,
29     "errors" : ""
30 }
```

O método **getinfo**, na linha 1 da Listagem 3.3, apresenta as informações relevantes sobre o atual estado da rede. É possível observar, linha 11, o IP interno da máquina

seguido pela porta utilizada para a conexão, outra informação relevante é sobre as carteiras existentes na rede para realizarem transações. Na Listagem 3.3 é observado na linha 19 a quantidade de blocos que foram minerados e também na linha 21 a quantidade de nós conectados a rede.

A partir da Listagem 3.3 observa-se a existência de duas carteiras na rede, que interagem entre si. A VM-1 e VM-6 possuem propriedades sobre estas carteiras e como experimento de funcionamento foi realizada uma transação da carteira da VM-6 para a carteira da VM-1 que pode ser identificada na Listagem 3.4.

Listagem 3.4: Informações de verificação de uma transação realizada

```
1  {"method":"getwallettransaction","params":["21a38f4784ad02636ce421369f8af805b929
       92aad6edeca44771e711b8553d59"],"id":1,"chain_name":"chain1"}
2      {
3          "balance" : {
4              "amount" : 0.00000000,
5              "assets" : [
6                  {
7                      "name" : "assets1",
8                      "assetref" : "71-265-5957",
9                      "qty" : -300.00000000
10                 }
11             ]
12         },
13         "myaddresses" : [
14             "1JBQVuzEZRzQhJwUuNARwc2oA2KTX82189iokZ"
15             ],
16         "addresses" : [
17             "1T5wAoTzv5TzZFnLdDPgWeU7DWci2HiBWABMwm"
18         ],
19         "permissions" : [
20         ],
21         "items" : [
22         ],
23         "data" : [
24         ],
25         "confirmations" : 505,
26         "blockhash" : "00418a75ebb5723b484b8dca00e9373983a953f3fdb7d0ee1b4a09946
               742fefd",
27         "blockindex" : 1,
28         "blocktime" : 1558627823,
29         "txid" : "21a38f4784ad02636ce421369f8af805b92992aad6edeca44771e711b8553d
               59",
30         "valid" : true,
31         "time" : 1558627808,
32         "timereceived" : 1558627808
33 }
```

Na Listagem 3.4 é possível observar informações da transação de VM-6 para VM-1. As linhas 15 e 17 visualiza-se os endereços das carteiras e também é possível observar informações como quantidade de confirmações, valor e data que a transferência ocorre e também outras informações relevantes para garantir a contabilidade da rede.

Listagem 3.5: Blocos que foram minerados e inseridos pela rede Blockchain

```
1      {
2          "hash" : "00d27a5d98675d6eb217160bc88f678e2f647e69fdbf3c5c46a5d8ed171e74
               3c",
3          "miner" : "1T5wAoTzv5TzZFnLdDPgWeU7DWci2HiBWABMwm",
4          "confirmations" : 185,
5          "height" : 828,
6          "time" : 1558903615,
7          "txcount" : 1
```



```
8       },
9       {
10          "hash" : "00d7f0c124e7cb850c92eb305a2f8ea975e9d776f10f9fe825073a2e97eaa2
                7d",
11          "miner" : "1JBQVuzEZRzQhJwUuNARwc2oA2KTX82189iokZ",
12          "confirmations" : 184,
13          "height" : 829,
14          "time" : 1558903615,
15          "txcount" : 1
16      },
17      {
18          "hash" : "00435d2ce2e2c426c63197adce3d3a046da2e6f6a1cc5da4a5c8ea47e51fdf
                89",
19          "miner" : "1T5wAoTzv5TzZFnLdDPgWeU7DWci2HiBWABMwm",
20          "confirmations" : 183,
21          "height" : 830,
22          "time" : 1558903646,
23          "txcount" : 1
24      },
25      {
26          "hash" : "0031338db14d866b306d451c195ce4067adc0dde5dfeedb124239627e753
                bdde",
27          "miner" : "1bABXfY7cshesG8d1UwTZdxiM1ehT9yY6Zh4Jw",
28          "confirmations" : 182,
29          "height" : 831,
30          "time" : 1558903654,
31          "txcount" : 4
32  }
```

A Listagem 3.5 possibilita a identificação e funcionamento quanto a mineração dos blocos realizados pela rede Blockchain. O Blockchain Multichain utiliza o mecanismo de consenso PBFT e como algoritmo de escalonamento de prioridades o sistema utiliza o *round-robin*. Uma outra averiguação que a Listagem 3.5 permite fazer é sobre o funcionamento do mecanismo, em que apresenta diferentes mineradores em sequência, observados no código pelas linhas 3, 11, 19 e 27, na mineração de novos blocos para a rede.

Com as informações apresentadas é possível averiguar o estado de funcionamento da arquitetura e comportamento da rede, sendo esta uma rede Blockchain com modelo privado e parcialmente descentralizada. A rede é composta por seis nós, sendo um deles o administrador, que delimita as permissões para cada uma destas VMs. Quanto ao sistema, é necessário avaliar seu desempenho a partir da realização de um ataque DoS em sua rede, com objetivo de avaliar o comprometimento da rede durante e após o ataque.

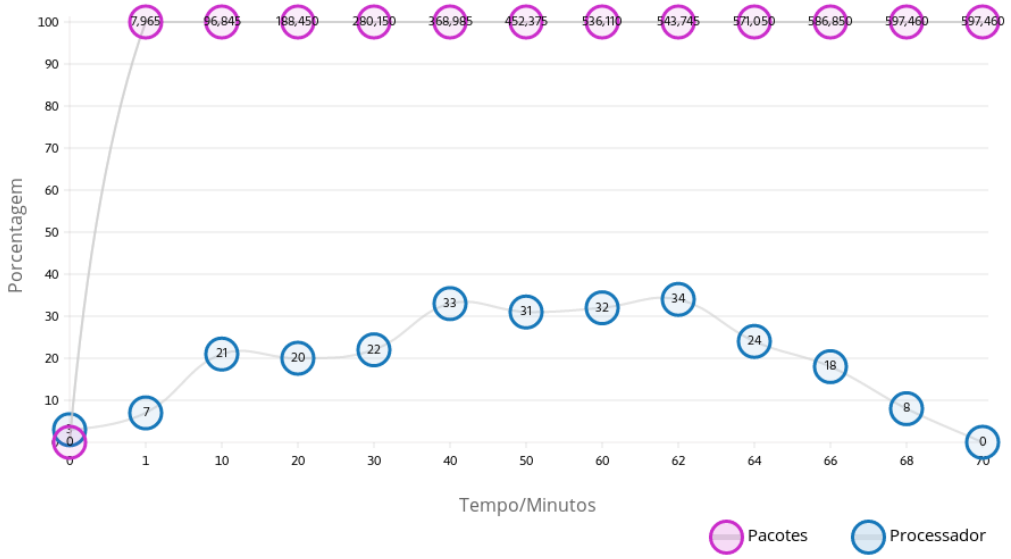
3.5.2.1. Ocorrência do Ataque de Negação de Serviço

A solução de Blockchain Multichain utiliza o protocolo de rede TCP. Como aplicação destes ataques foram escolhidos os ataques: *SSH Flood* e o *Transaction Flood*.

O ataque SSH foi realizado a partir da VM-6 endereçado para a VM-1. A partir deste ataque foi observado que não causava nenhum dano na VM-1 da rede Blockchain Multichain. Não foram identificados alterações de processamento e nem do uso de memória que causassem impactos negativos para o nó ou propriamente para a rede. O principal motivo pela não ocorrência de nenhum efeito, é a necessidade de aplicação de cargas no ataque.

Quanto ao ataque *Transaction Flood* foi realizado a partir da carteira VM-6 que realizava transações para a carteira pertencente a VM-1. O mecanismo de consenso PBFT é um mecanismo com baixo consumo de CPU, ou seja, não necessita de considerável uso de processamento para validação de transações ou criação de novos blocos. Este comportamento pode ser observado a partir da Figura 3.13 durante o período que a VM recebeu os ataques gerados pela VM-6.

Figura 3.13: Análise do consumo de processador durante a o ataque *Transactions Flood*.



A partir da Figura 3.13 e Tabela 3.10 é possível observar um baixo crescimento no uso de CPU durante o processo de ataque. Na Figura 3.13 possui alguns pontos em que há pequenas alterações durante a execução do ataque, isso deve-se ao fato de ocorrer retransmissões TCP que foram captadas durante a execução do ataque.

Tabela 3.10: Tabela do consumo de processador durante o ataque *Transaction Flood*.

Tempo/Min	0	1	10	20	30	40	50	60	62	64	66	68	70
Processador	3,3%	7,25%	21,76%	20,23%	22,94%	33,00%	31,09%	32,69%	34,08%	24,40%	18,25%	8,07%	0,9%
Pacotes	0	7965	96845	188450	280150	368985	452375	536110	543745	571050	586850	597460	597460

Na Figura 3.14 e Tabela 3.11 observa-se um expressivo impacto do ataque no uso da memória da VM. A partir da Figura 3.14 que a ponto que novas transações eram criadas havia um crescimento significante na quantidade de memória que era consumida pela rede Blockchain. Há alguns pontos em que a memória subitamente reduzia a taxa de uso, existe a necessidade de uma análise mais profunda para compreender o motivo desta redução, uma possível explicação seria o descarte em blocos realizado pela plataforma.

Um ponto relevante é que após uma hora de ataque o sistema apresentou estouro/-falta de memória. A falta de memória implica no encerramento abrupto da aplicação e a partir deste encerramento foi percebido que do montante de transações enviadas entorno de 10% à 15% foram validadas pela aplicação, sendo as demais foram perdidas.

Figura 3.14: Análise do consumo de memória durante o ataque *Transactions Flood*.

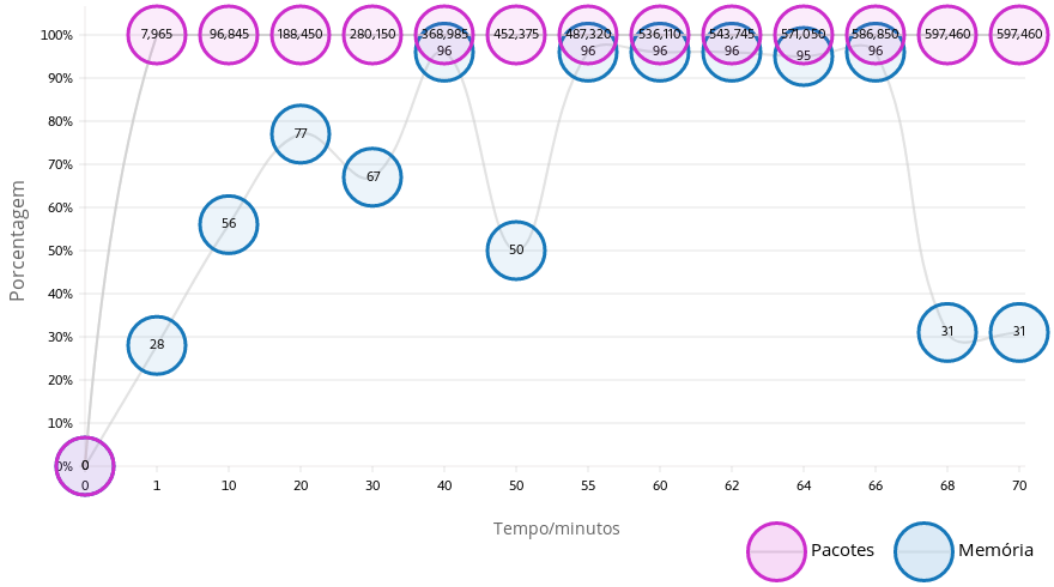


Tabela 3.11: Tabela do consumo de memória durante o ataque *Transaction Flood*.

Tempo/Min	0	1	10	20	30	40	50	55	60	64	66	68	70
Memória	0%	28,65%	56,28%	77,06%	67,09%	96,14%	50,57%	96,16%	96,18%	95,84%	96,72%	31,15%	31,15%
Pacotes	0	7965	96845	188450	280150	368985	452375	487320	536110 §	571050	586850	597460	597460

Na Figura 3.15 é observado o tráfego TCP durante o período da realização do ataque *Transactions Flood*. É possível verificar, a partir dos dados da Tabela 3.12, que a aplicação estava enviando muito mais dados que recebendo, demonstrando atrasos na rede Blockchain e também perda de transações e pacotes.

Figura 3.15: Análise de tráfego TCP durante o ataque *Transaction Flood*.

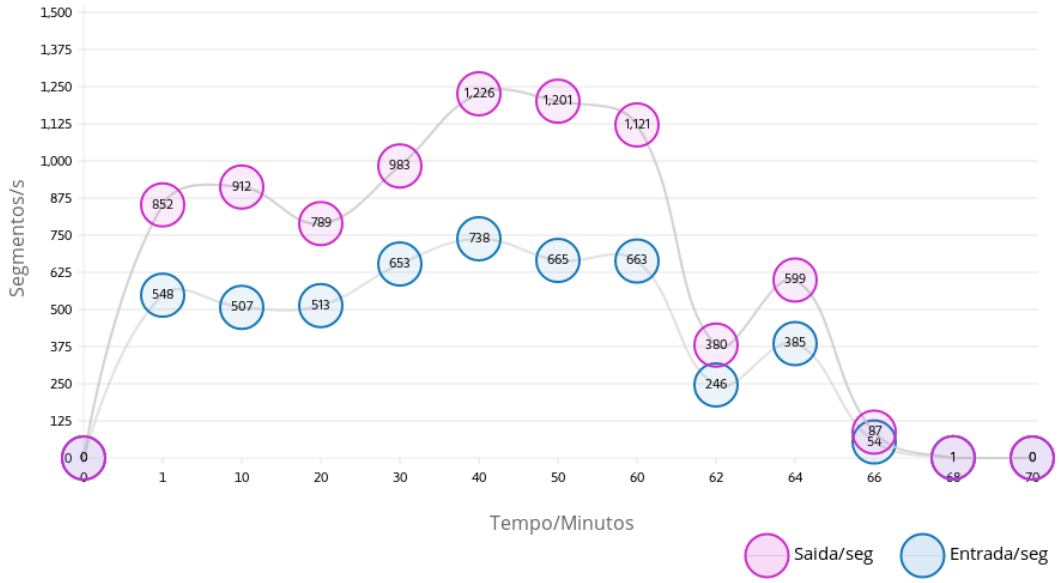


Tabela 3.12: Tabela do tráfego TCP durante o ataque *Transaction Flood*.

Tempo/Min	0	1	10	20	30	40	50	55	60	64	66	68	70
Entrada/seg	0	548,78	507,50	513,38	653,85	738,33	665,62	663,38	246,78	385,97	54,15	1,23	0,85
Saida/seg	0	852,33	912,08	789,08	983,45	1226,33	1201,48	1121,55	380,57	599,48	87,25	1,35	0,85

A partir deste experimento é observado as implicações do ataque *Transactions Flood* no desempenho da rede Blockchain. Este cenário indica uma fragilidade de uma rede parcialmente descentralizada e privada perante seus pares que também pertencem a mesma rede. Afinal, com apenas um atacante foi possível realizar estouro de memória no nó que administra a rede, causando instabilidade de modo geral na rede Blockchain

3.5.3. Análises dos resultados

Com base na experimentação realizada, é possível observar várias potenciais vulnerabilidades e ameaças que podem ser exploradas nas redes tipo Blockchain. Os resultados desta análise são condizentes com as necessidades que foram abordadas por [50]. O trabalho apresentam a questão de Ataques de Negação de Serviço como um ataque que além de causar danos para a rede de Blockchain também influencia problemas em sistemas que podem ser aplicados conjuntamente. Contudo, para estas vulnerabilidades existe a possibilidade de mitigação dos impactos dos ataques através de mecanismos de segurança e flexibilização na configuração do sistema. Para consolidação dos resultados obtidos, a Tabela 3.13 apresenta a relação entre os critérios analisados, os problemas encontrados relacionados a este critérios e possíveis boas práticas para minimizar os problemas.

Tabela 3.13: Visão geral dos resultados da análise do trabalho.

Critérios	**Consenso**	**Problemas**	**Boas práticas**
Procedência dos dados	PoW	Não apresentou problemas em relação a segurança dos dados	Realizar verificações de transações e blocos para garantia contínua da seguridade
	PBFT	Problemas com validação de Transações, Possibilidade de alteração do conteúdo das transações	Monitoramento CPU/Memória
Controle e Gerenciamento do Nó	PoW	Problemas com Mineração, Tráfego intenso em seus protocolos	Monitoramento de tráfego interno, Monitoramento de processamento
	PBFT	Problemas com administração da rede Blockchain	Monitoramento de transações

Para o critério de procedência dos dados foram encontrados problemas relacionados a validação de transações e blocos, gerados por dificuldades de mineração ou queda do sistema. Quanto a esta procedência dos dados, existe uma outra questão relevante apresentadas por Blockchain privados que é a questão do anonimato. Esta vulnerabilidade é conhecida como Privacidade de Transações, é considerada uma vulnerabilidade para uma rede Blockchain, mas é interessante para instituições que desejam auditar seu sistema e detectar possíveis causadores de problemas em sua rede. Uma possível solução encontrada para ambos os casos, é a realização de monitoramentos de CPU e memória e também monitoramento de violações da rede interna, que causam danos mais graves ao funcionamento do sistema.

No critério de controle e gerenciamento do nó, os principais problemas encontrados foram relacionados a mineração, tráfego intenso em suas redes internas e por parte do Blockchain parcialmente descentralizado a questão de administração da rede Blockchain. Estes problemas relacionados ao controle e gerenciamento dos nós abrem portas para diversas outras vulnerabilidades, principalmente vinculadas a rede e poder computacional, Seção 3.2. Como possível boa prática para evitar que estas vulnerabilidades sejam exploradas é interessante que haja monitoramento de tráfego, transações e de processamento.

3.6. Considerações

Em relação às versões, modelos, mecanismos de consenso do Blockchain sobressai-se a preocupação em apresentar as principais características envolvidas. A partir desta preocupação, a realização de comparações entre os modelos, algoritmos de consenso, versões entre outras características para maior entendimento das diversas possibilidades de combinações para aplicações do Blockchain. Sobre as características, foram apresentadas sistematicamente os principais meios, ou plataformas, em que a tecnologia Blockchain se faz presente sua utilização. além disso, são citadas as três plataformas mais utilizadas no uso de contratos inteligentes e aplicativos descentralizados que são: Ethereum, Hyperledger e o MultiChain. Em cada uma destas plataformas foram elencadas características da aplicação e realizado comparações entre elas, para ficar mais claro as necessidades e conhecimentos necessários para uso das mesmas.

Relacionado a preocupação com a segurança do Blockchain, o estudo apresentado inclui uma proposta de classificação das principais ataques e vulnerabilidades podem afetar as implementações do conceito de Blockchain. Neste estudo foram elencadas algumas vulnerabilidades do Blockchain que são: DoS, Ataque Eclipse, Ataque de Vivacidade, Gasto Duplo, Mineração Egoísta, Otimização de Contratos Inteligentes, Privacidade de Transações, Retenção de Blocos e Vulnerabilidade 51%. Devido a não ser identificado pelos autores um padrão para classificar estes ataques/vulnerabilidades, estes foram classificados em três grupos: Redes, Poder Computacional e Usuário.

De um modo geral, quanto ao uso dos mecanismos de consenso, é perceptível que há possibilidade, através das boas práticas, destes problemas serem minimizados. Contudo, por mais críticas que sejam as vulnerabilidades existentes, o uso da tecnologia Blockchain tem crescido e mostrado ao mercado que o investimento e desenvolvimento da tecnologia, assim como de seus métodos, pode proporcionar diversos benefícios às instituições envolvidas.

Agradecimentos

O presente trabalho foi em parte financiado pelo CNPq, através das Bolsas de Produtividade em Pesquisa 301198/2017-9 e Produtividade em Desenvolvimento Tecnológico e Extensão Inovadora 311667/2014-7.

Os autores agradecem o apoio do Laboratório de Arquitetura e Redes de Computadores (LARC) e Laboratório de Sustentabilidade (LASSU) do Departamento de Engenharia de Produção e Sistemas Digitais (PCS) da Escola Politécnica da Universidade de São Paulo (USP).

Os autores agradecem o apoio do Laboratório de Processamento Paralelo e Distribuído (LabP2D) no Centro de Ciências tecnológicas (CCT) da Universidade do Estado de Santa Catarina (UDESC).

Esse trabalho foi financiado com recursos da Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC).

Referências

[1] M. Swan, *Blockchain: Blueprint for a New Economy*. "O'Reilly Media, Inc.", Jan. 2015.

[2] P. Tasca and C. Tessone, "A taxonomy of blockchain technologies: Principles of identification and classification," *Ledger*, vol. 4, no. 0, 2019. [Online]. Available: http://www.ledgerjournal.org/ojs/index.php/ledger/article/view/140

[3] P. J. Taylor, T. Dargahi, A. Dehghantanha, R. M. Parizi, and K.-K. R. Choo, "A systematic literature review of blockchain cyber security," *Digital Communications and Networks*, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352864818301536

[4] B. Rodrigues, E. J. Scheid, R. Blum, T. Bocek, and B. Stiler, "Blockchain and Smart Contracts – From Theory to Practice," in *Tutorials of IEEE International Conference on Blockchain and Cryptocurrency*. Seoul, South Korea: IEEE Computer Society Press, May 2019, p. 31. [Online]. Available: https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/publications/CNSM18-Tutorial.pdf

[5] I.-C. Lin and T.-C. Liao, "Survey of blockchain security issues and challenges," *International Journal of Network Secutiry*, 2017.

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *International Journal of Network Secutiry*, 2008. [Online]. Available: "https://bitcoin.org/bitcoin.pdf"

[7] T. E. Foundation, "Ethereum homestead documentation," 2018.

[8] Hyperledger, "An introduction to Hyperledger," White Paper. Available: https://www.hyperledger.org/wp-content/uploads/2018/07/HL_Whitepaper_IntroductiontoHyperledger.pdf, 2018.

[9] G. Greenspan, "Multichain private blockchain – white paper," www.multichain.com/download/MultiChain-White-Paper.pdf, 2015.

[10] Plurasight, "Blockchain architecture," 2017. [Online]. Available: https://www.pluralsight.com/guides/blockchain-architecture

[11] NIST, *FIPS 180-4: Secure Hash Standard (SHS)*, National Institute of Standards and Technology, Gaithersburg, MD, USA, August 2015.

[12] R. C. Merkle, "Protocols for public key cryptosystems," in *IEEE Symposium on Security and Privacy*, April 1980, pp. 122–134.

[13] J. Garay and A. Kiayias, "SoK: A consensus taxonomy in the blockchain era," Cryptology ePrint Archive, Report 2018/754, 2018, https://eprint.iacr.org/2018/754.

[14] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '87. New York, NY, USA: ACM, 1987, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/41840.41841

[15] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *Trust and Trustworthy Computing*, M. Conti, M. Schunter, and I. Askoxylakis, Eds. Cham: Springer International Publishing, 2015, pp. 163–180.

[16] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proc. of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186.

[17] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 104–121.

[18] R. Patterson, "Alternatives for Proof of Work, Part 2: Proof of Activity, Proof of Burn, Proof of Capacity, and Byzantine Generals — Bytecoin Blog," Mar. 2016. [Online]. Available: https://web.archive.org/web/20160304055454/https://bytecoin.org/blog/proof-of-activity-proof-of-burn-proof-of-capacity/

[19] G. Kostarev, "Review of blockchain consensus mechanisms," Jul. 2017. [Online]. Available: https://blog.wavesplatform.com/review-of-blockchain-consensus-mechanisms-f575afae38f2

[20] J. Kwon, "Tendermint: Consensus without mining - v0.6," White Paper. Available: http://docplayer.net/50173080-Tendermint-consensus-without-mining.html, 2014.

[21] V. Buterin, "On public and private blockchains," 2015. [Online]. Available: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/

[22] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: a survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[23] namecoin, "Name coin," 2018. [Online]. Available: https://namecoin.org/

[24] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies: Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*. Berlin, Heidelberg: Springer, 2001, pp. 46–66.

[25] J. Benet, "IPFS: content addressed, versioned, P2P file system," arXiv preprint arXiv:1407.3561, 2014, see also https://ipfs.io.

[26] E. Larcheveque, A. Caswell, and A. Ferron, "BitID: Bitcoin authentication open protocol," https://github.com/bitid/bitid, 2016.

[27] G. Wood, "Polkdaot: Vision for a Heterogeneous Multi-Chain Framework," November 2016, Accessed: 2019-07-04. [Online]. Available: https://polkadot.network/PolkaDotPaper.pdf

[28] Aelf, "aelf - A Multi-Chain Parallel Computing Blockchain Framework," June 2018, Accessed: 2019-07-04. [Online]. Available: https://aelf.io/gridcn/aelf_whitepaper_EN.pdf?v=1.6

[29] F. Greve, L. Sampaio, J. Abijaude, A. A. Coutinho, I. Brito, and S. Queiroz, *Blockchain e a Revolução do Consenso sob Demanda*. Sociedade Brasileira de Computação (SBC), 2018, ch. Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC).

[30] B. Curran, "What is delegated proof of stake consensus? (DPoS) complete beginner's guide," 2018.

[31] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, October 2017.

[32] BitCoin.org, "How does Bitcoin work?" Bitcoin Official website: https://bitcoin.org/en/how-it-works, 2018.

[33] G. Greenspan, "(white paper) decentralized financial system – Credits – v 2.1," Credits, Tech. Rep., 2018, available: credits.com/Content/Docs/TechnicalWhitePaperCREDITSEng.pdf.

[34] BR, "Ethereum white paper made simple," Blockchain Review, Tech. Rep., 2018, available: https://cryptoverze.com/wp-content/uploads/2018/11/02.01._final_Ethereum-White-Paper-Made-Simple.pdf.

[35] V. Buterin, "Ethereum white paper – a next generation smart contract & decentralized application platform," Ethereum.org, Tech. Rep., 2018, available: http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

[36] G. Greenspan, "Multichain private blockchain," 2018.

[37] B. Rodrigues, T. Bocek, and B. Stiller, "Enabling a cooperative , multi-domain DDoS defense by a Blockchain signaling system (BloSS)," in *Proc. of the 42nd IEEE Conference on Local Computer Networks 2017 (LCN) – Demos*, 2017, pp. 1–3.

[38] A. Singh, T. Ngan, P. Druschel, and D. S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, Apr. 2006, pp. 1–12.

[39] D. K. Tosh, S. Shetty, X. Liang, C. A. Kamhoua, K. A. Kwiat, and L. Njilla, "Security implications of Blockchain cloud with analysis of block withholding attack," in *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.

[40] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, Aug. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X17318332

[41] E. Le Jamtel, "Swimming in the monero pools," in *2018 11th International Conference on IT Security Incident Management IT Forensics (IMF)*, May 2018, pp. 110–114.

[42] I. Eyal, "The Miner's Dilemma," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 89–103.

[43] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *CoRR*, vol. abs/1311.0243, 2013. [Online]. Available: http://arxiv.org/abs/1311.0243

[44] A. Kiayias and G. Panagiotakos, "On trees, chains and fast transactions in the blockchain," Cryptology ePrint Archive, Report 2016/545, 2016, https://eprint.iacr.org/2016/545.

[45] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-Optimized Smart Contracts Devour Your Money," *arXiv:1703.03994 [cs]*, Mar. 2017, arXiv: 1703.03994. [Online]. Available: http://arxiv.org/abs/1703.03994

[46] P. Mell and T. Grance, "(SP 800-145) the NIST definition of cloud computing," National Institute of Standards & Technology, Gaithersburg, MD, United States, Tech. Rep., 2011.

[47] S. Shetty, V. Red, C. Kamhoua, K. Kwiat, and L. Njilla, "Data provenance assurance in the cloud using blockchain," in *Disruptive Technologies in Sensors and Sensor Systems*, vol. 10206. International Society for Optics and Photonics, May 2017, p. 102060I. [Online]. Available: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10206/102060I/Data-provenance-assurance-in-the-cloud-using-blockchain/10.1117/12.2266994.short

[48] S. Report, "Funcionários são responsáveis por nove em cada dez violações de dados na nuvem," May 2019. [Online]. Available: http://www.securityreport.com.br/overview/funcionarios-sao-responsaveis-por-nove-em-cada-dez-violacoes-de-dados-na-nuvem/

[49] G. Zyskind, O. Nathan, and A. . Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*, May 2015, pp. 180–184.

[50] A. Rot and B. Blaicke, "Blockchain's future role in cybersecurity. analysis of defensive and offensive potential leveraging blockchain-based platforms," in *9th International Conference on Advanced Computer Information Technologies (ACIT)*, 2019.

[51] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982, availble: people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf.

Capítulo 4

Introdução à Engenharia Reversa de Aplicações Maliciosas em Ambientes Linux

Marcus Botacin[1], Lucas Galante[2], Otávio Silva[2], and Paulo Lício de Geus[2]

[1]Universidade Federal do Paraná (UFPR)
[2]Universidade Estadual de Campinas (UNICAMP)

Abstract

Malicious software have been evolving to look even more indubious to the targeted system. Moreover, they have been relying on obfuscation and anti-analysis techniques to avoid their behavior being discovered during their execution, thus making forensic procedures harder to be conducted on a proper manner. Reverse engineering is an useful procedure to handle such type of threat, pinpointing the best inspection paths to be followed by analysts, such as how to unpack a given sample or which protection techniques are implemented by a sample. In this course, we present reverse engineering techniques to analyze malicious software samples, thus introducing the reader to general malware handling procedures. The course introduces techniques both in userland as well as in the OS kernel, including dynamic application tracing, debugging and rootkit techniques.

Resumo

Aplicações maliciosas têm evoluído de forma a cada vez mais permanecerem ocultas no sistema alvo. Além disso, essas aplicações utilizam de técnicas de ofuscação e/ou anti-análise a fim de evitar a descoberta de seu comportamento durante a execução, impedindo que atividades de perícia forense computacional sejam realizadas adequadamente. A engenharia reversa é um processo útil na análise desse tipo de aplicação, uma vez que pode-se descobrir quais caminhos seguir em uma perícia, por exemplo, como desempacotar a aplicação ou que tipo de truques ela usa para evitar ser destrinchada. Neste minicurso, apresentamos técnicas de engenharia reversa para análise de aplicações maliciosas tomando como base o ambiente Linux, visando introduzir os participantes ao tema e capacitá-los no tratamento de aplicações maliciosas em geral. O curso aborda técnicas tanto em modo usuário quanto em modo kernel do sistema operacional, incluindo o traço dinâmico de aplicações, técnicas de depuração e de rootkits.

4.1. Introdução

A plataforma `Linux` tem se consolidado como uma das principais plataformas para o desenvolvimento de novas tecnologias no mundo contemporâneo. Atualmente, ela suporta desde ambientes de computação em nuvem (*cloud*) [Venezla 2012] até o sistema operacional para *smartphones* `Android` [Android 2017]. Contudo, ao mesmo tempo em que proporciona o desenvolvimento de novas tecnologias, a plataforma `Linux` torna-se alvo de ataques, passando a requerer o desenvolvimento de ferramentas de análise de binários [Afonso et al. 2015] e para procedimentos forenses [Vecchia and Coral 2014].

Um dos principais vetores de ataque contra a plataforma `Linux` são códigos maliciosos (*malicious software* ou *malware*), que são arquivos executáveis que apresentam deliberadamente intenções prejudiciais à sistemas computacionais [Skoudis and Zeltser 2003], o que pode incluir do vazamento de informações sensíveis do usuário ao controle total do sistema infectado. Na prática, ataques por *malware* contra a plataforma `Linux` são frequentes e causam prejuízos significativos. Em 2016, a *botnet* `Mirai` [Fruhlinger 2018] causou interrupções na conexão à Internet da costa Atlântica dos Estados Unidos. Em 2017, o *ransomware* `Erebus` [Z. Chang 2017] sequestrou os servidores de um provedor de conteúdo sul-coreano em troca de resgate.

Para se prevenir e combater ataques por *malware*, diversas técnicas de análise são empregadas, dentre as quais se destaca a engenharia reversa dos códigos maliciosos. A engenharia reversa consiste em inspecionar o binário malicioso de modo a compreender seu funcionamento, identificar seu alvo no sistema infectado e potencialmente criar mecanismos para a defesa contra este tipo de ameaça. As técnicas de engenharia reversa apresentam muitas estratégias comuns a diferentes plataformas, como a execução do binário desconhecido em um ambiente controlado (*sandbox*). Entretanto, diferentes plataformas apresentam especificidades quanto ao alvo da execução em *sandbox*. Como exemplo, enquanto na plataforma `Windows` exemplares de *malware* afetam o subsistema de chaves de registro [Botacin et al. 2018], na plataforma `Linux`, exemplares de *malware* afetam o subsistema de arquivo `/proc` [Galante et al. 2018], de modo que entender as particularidades das técnicas de engenharia reversa em cada plataforma é essencial para prover respostas adequadas aos ataques que estas sofrem.

Contudo, apesar dos impactos da infecção por *malware* na plataforma `Linux` serem reais e das técnicas de engenharia reversa poderem auxiliar no combate à este ataques, a literatura acadêmica tem dado pouca atenção a plataforma `Linux` e focado na plataforma `Windows`, dada sua concentração de mercado historicamente grande, até a ascensão recente da plataforma `Linux`. Frente a este cenário de uso crescente e técnicas de análise pouco difundidas em comparação a plataforma anteriormente dominante, vislumbramos a necessidade de se apresentar os conceitos de engenharia reversa com foco específico nesta plataforma, fato que motiva o desenvolvimento deste curso.

Objetivos. Este minicurso busca ser uma introdução à engenharia reversa de aplicações em ambientes `Linux`, com foco na análise de aplicações maliciosas, em especial as que implementam técnicas de evasão de procedimentos de análise. Durante o minicurso, os participantes desenvolverão habilidades básicas de engenharia reversa, como a caracterização de binários, a identificação das proteções usadas e alguns contornos (*bypass*) para tais proteções.

Sobre o curso. Este curso é proposto na modalidade majoritariamente prática, através de exercícios de análise de binário reais, mas contando com o devido suporte teórico para embasar as decisões tomadas pelos participantes. O curso ocorrerá no formato de uma competição `capture-the-flag`, na qual os participantes serão desafiados a resolver alguns desafios que levem à engenharia reversa do binário dado. Desta forma, este capítulo corresponde a parte teórica do curso, fornecendo subsídios para que os alunos realizem os desafios.

Audiência. Este curso se destina ao público que pretende iniciar seus estudos na área de análise de binários. Não são exigidos conhecimentos prévios na área de segurança, apenas familiaridade com o uso de sistemas `GNU/Linux` e seu terminal (`bash`), e conhecimentos básicos em alguma linguagem de programação. Acreditamos que o curso possa ser de interesse também para quem já possua familiaridade com as soluções de análise da plataforma `Linux`, como *debuggers*, dado que as técnicas de engenharia reversa são distintas das empregadas comumente na depuração de aplicações convencionais.

Materiais. Todo os binários a serem analisados durante o curso podem ser obtidos através do repositório: `https://github.com/marcusbotacin/Malware.Reverse.Intro`, de modo que os leitores possam acompanhar este capítulo mesmo após a realização do curso presencial. Todos os códigos são disponibilizados juntamente com arquivos de compilação (`Makefiles`), não requerendo que os participantes tenham conhecimentos avançados, como a compilação de *drivers* de *kernel* para executar as tarefas propostas.

Estrutura. Este capítulo é dividido da seguinte forma: Na Seção 4.2, apresentamos a estrutura básica dos binários ELF utilizados nas plataformas `Linux` e ferramentas básicas para a manipulação destes; Na Seção 4.3, introduzimos os primeiros conceitos de engenharia reversa e apresentamos as verificações iniciais a serem realizadas sobre arquivos binários sem a execução do mesmo (análises estáticas), como a identificação do tipo de arquivo inspecionado, a presença de bibliotecas, e sinais de ofuscação de código, de modo a se obter uma compreensão inicial dos possíveis comportamentos exibidos pelo binário; Na Seção 4.4, estendemos as análise de observações passivas para modificações ativas no binário (*binary patching*), de modo a contornar verificações implementadas pelas rotinas de anti-análise; Na Seção 4.5, estendemos as análises anteriormente descritas através da execução dos binários desconhecidos em um ambiente controlado (*sandbox*), permitindo, assim, a análise de exemplares ofuscados; Na Seção 4.6, tal qual realizado para as técnicas de análise estática, estendemos as técnicas dinâmicas de observações passivas para interferências ativas na execução dinâmica de código, permitindo, assim, contornar técnicas de anti-análise implementadas pelos binários; Apresentamos, também, como os atacantes se utilizam destas mesmas técnicas para implementar seus comportamentos evasivos, tais como *rootkits*; Na Seção 4.7, apresentamos abordagens complementares de análise, com focos em subsistemas específicos, como a monitoração do subsistema de arquivos para recuperação de arquivos deletados pelos exemplares maliciosos; Na Seção 4.8, estendemos os conceitos e soluções anteriormente apresentadas em modo usuário (*userland*) para o modo mais privilegiado (*kernel*), visando contornar mecanismos de anti-análise mais sofisticados; Na Seção 4.9, apresentamos conceitos básicos de monitoração de tráfego de rede; Na Seção 4.10, apresentamos nossas considerações finais.

4.2. Binários ELF: Definição e Manipulação

Diferentes plataformas representam seus aplicativos de diferentes maneiras, de modo que compreender a representação de um arquivo executável é essencial para a execução de procedimentos de investigação de binários desconhecidos e suspeitos. Nesta seção, introduzimos o tipo binário utilizado pelas plataformas `Linux` e ferramentas básicas para a manipulação destes.

4.2.1. A Estrutura dos Binários ELF

O *Executable and Linkable Format* (ELF) é o formato padrão de arquivos executáveis na plataforma `Linux` [O'Neill 2016] e estabelece uma estrutura a ser seguida por todos as aplicações da plataforma de modo que o sistema tome conhecimento, através dos campos padronizados, de como carregar o conteúdo do arquivo binário em memória e inicializar um processo. Um arquivo ELF, como definido em `elf.h`, possui a estrutura básica apresentada na Figura 4.1.

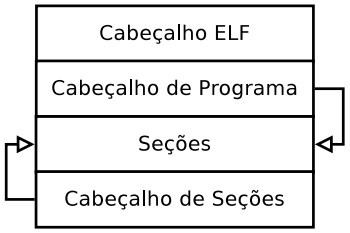


Figura 4.1. Arquivo ELF. Estrutura Básica.

O cabeçalho ELF, definido como mostrado no Código 4.1, é uma estrutura obrigatória e está sempre presente no deslocamento (*offset*) "0" de todo arquivo ELF. Ele apresenta as informações básicas do binário, como seu tipo (`e_type`)–podendo ser binários autocontidos ou bibliotecas—, a arquitetura do sistema alvo (`e_machine`), entre outras informações essenciais a execução.

```
1  #define EI_NIDENT 16
2  typedef struct {
3         unsigned char e_ident[EI_NIDENT];
4         uint16_t      e_type;
5         uint16_t      e_machine;
6         uint32_t      e_version;
7         ElfN_Addr     e_entry;
8         ElfN_Off      e_phoff;
9         ElfN_Off      e_shoff;
10        uint32_t      e_flags;
11        uint16_t      e_ehsize;
12        uint16_t      e_phentsize;
13        uint16_t      e_phnum;
14        uint16_t      e_shentsize;
15        uint16_t      e_shnum;
16        uint16_t      e_shstrndx;
17    } ElfN_Ehdr;
```

Código 4.1. Definição do cabeçalho de um arquivo ELF.

O cabeçalho de programa, tal qual definido no Código 4.2, armazena informações básicas de como as informações contidas no arquivo devem ser interpretadas (`p_type`), em qual posição as informações se localizam (`*_addr`) e qual o tamanho dos blocos contendo estas informações (`*sz`).

```
1  typedef struct {
2         uint32_t   p_type;
3         uint32_t   p_flags;
4         Elf64_Off  p_offset;
5         Elf64_Addr p_vaddr;
6         Elf64_Addr p_paddr;
7         uint64_t   p_filesz;
8         uint64_t   p_memsz;
9         uint64_t   p_align;
10     } Elf64_Phdr;
```

Código 4.2. Definição do cabeçalho de programa de arquivo ELF.

Dentre os tipos que um arquivo ELF pode assumir, tal qual apontado pelo cabeçalho de programa, estão binários autocontidos e arquivos objetos, utilizado para a implementação de códigos em bibliotecas. A ligação de bibliotecas à binários pode ser realizada de duas maneiras [Simmonds 2015]: (i) estaticamente, em tempo de compilação (`gcc -static <arquivo>`); ou (ii) dinamicamente, em tempo de execução. Cada uma destas formas de ligação apresenta vantagens e desvantagens: Por um lado, ligações estáticas, por embutirem no binário todo o código necessário para a execução da aplicação, permitem que o binário seja executado em diversos sistemas sem a necessidade de se instalar qualquer dependência. Por outro lado, estas aumentam significativamente o tamanho final do binário gerado em comparação com binários ligados à bibliotecas dinâmicas.

O cabeçalho de seção de um arquivo ELF, como mostrado no Código 4.3, estabelece como o conteúdo da aplicação está disposto dentro do binário. Cada tipo de dado (instruções de máquina, variáveis estáticas, entre outros) é disposta em uma seção diferente, cada uma recebendo um nome (`sh_name`) e atributos (e.g., `sh_flags`) próprios.

```
1  typedef struct {
2         uint32_t   sh_name;
3         uint32_t   sh_type;
4         uint64_t   sh_flags;
5         Elf64_Addr sh_addr;
6         Elf64_Off  sh_offset;
7         uint64_t   sh_size;
8         uint32_t   sh_link;
9         uint32_t   sh_info;
10        uint64_t   sh_addralign;
11        uint64_t   sh_entsize;
12    } Elf64_Shdr;
```

Código 4.3. Definição do cabeçalho de seção de um arquivo ELF.

O Código 4.4 exibe as seções de um um binário ELF típico: Instruções de máquina são armazenadas na seção `.text`; Variáveis inicializadas com e sem permissão de escrita são armazenados, respectivamente, nas seções `.data` e `.rodata`; Variáveis e outros dados dinâmicos são armazenados nas seções `*dyn`; Bibliotecas dinâmicas e outros apontadores carregados em tempo de execução são armazenados na região correspondente as seções do tipo `*symbols*`.

```
[Nr] Nome
[ 1] .interp            [ 2] .note.ABI-tag
[ 3] .note.gnu.build-i  [ 4] .gnu.hash
[ 5] .dynsym            [ 6] .dynstr
[ 7] .gnu.version       [ 8] .gnu.version_r
[ 9] .rela.dyn          [10] .rela.plt
[11] .init              [12] .plt
[13] .plt.got           [14] .text
[15] .fini              [16] .rodata
[17] .eh_frame_hdr      [18] .eh_frame
[19] .init_array        [20] .fini_array
[21] .jcr               [22] .dynamic
[23] .got               [24] .got.plt
[25] .data              [26] .bss
[27] .comment           [28] .shstrtab
[29] .symtab            [30] .strtab
```

Código 4.4. Exibição das seções de um arquivo ELF.

4.2.2. Manipulação Básica de Binários ELF

Uma vez que se compreenda a estrutura básica dos binários ELF, pode-se proceder a inspeção das seções e cabeçalhos dos binários de modo a identificar construções suspeitas. A plataforma `Linux` possui diversas soluções para a manipulação de binários ELF, incluindo a manipulação direta através dos cabeçalhos disponíveis para a linguagem C. Contudo, nesta seção, para fins didáticos, exemplificaremos a manipulação de binários ELF através da solução `pyelftools` [eliben 2017].

O Código 4.5 ilustra como listar o nome das seções de um binário ELF via `pyelftools`. Em análise de *malware*, este tipo de verificação é essencial para se identificar nomes de funções suspeitas, como os nomes deixados pela solução de empacotamento e ofuscação `UPX` (detalhado na seção 4.3). `Pyelftools` também permite ao analista verificar o tamanho das seções, cabeçalhos e valores dos apontadores. Este tipo de informação é útil pois muitos binários maliciosos acabam sendo malformados devido as rotinas de ofuscação e apresentam valores nulos nestes campos, o que pode servir como um indicador de infecção ou suspeição.

```
def process_file(filename):
    with open(filename, 'rb') as f:
        elffile = ELFFile(f)
        for section in elffile.iter_sections():
            print(section.name)
```

Código 4.5. Enumeração das seções de um arquivo ELF através da solução pyelftools.

4.3. Análise Estática

No contexto de engenharia reversa de aplicações maliciosas, referimos a análise estática como o conjunto de métodos utilizados para obter informações sobre os arquivos binários suspeitos sem a efetiva execução destes [Sikorski and Honig 2012]. No contexto de análise forense, este tipo de análise é usualmente a etapa inicial de um conjunto de procedimentos complexos realizados com o objetivo de se obter informações preliminares sobre o artefato em análise.

Através dos procedimentos de análise estática, um analista pode identificar o tipo do arquivo em análise e se bibliotecas de código são ligadas à este, permitindo a classificação do artefato. Um analista pode também inspecionar se o arquivo apresenta *strings* que possam prover informações forenses. Finalmente, um analista pode obter o código *assembly* para identificar, preliminarmente, possíveis comportamentos maliciosos ou técnicas de evasão. Apresentamos, a seguir, soluções e procedimentos para realizar estes passos de análise.

4.3.1. O utilitário e comando `file`

O comando *file* é frequentemente utilizado para o reconhecimento do tipo de dado contido em um arquivo. O Código 4.6 exemplifica a saída do comando para diferentes arquivos de entrada: um código na linguagem C, um diretório do sistema, e um arquivo `pdf`. É importante notar que a identificação do formato de arquivo pelo comando `file` é dado por uma base de dados própria [GNU 2018], sendo independente da extensão utilizada para nomear o arquivo.

```
1 >$ file hello.c
2 hello.c: ASCII text
3 >$ file Documents/
4 Documents/: directory
5 >$ file wticg.pdf
6 wticg.pdf: PDF document, version 1.4
```

Código 4.6. Comando file em diferentes arquivos.

O comando `file` é de especial utilidade para procedimentos de análise quando aplicados a arquivos ELF, como ilustrado pelo Código 4.7. Através deste, pode-se identificar, por exemplo: (i) que o arquivo é um binário executável (`ELF`) para arquiteturas de 64 *bits* (`x86-64`), como mostrado na linha 2; e que (ii) o arquivo se liga a bibliotecas de forma dinâmica, como mostrado na linha 3. A linha 4 exibe o *hash* (identificador único) do arquivo.

```
1 >$ file dynamic-malicious.bin
2 dynamic-malicious.bin: ELF 64-bit LSB executable, x86-64, version 1 (
    SYSV),
3 dynamically linked, interpreter /lib64/l, for GNU/Linux 2.6.32,
4 BuildID[sha1]=5e42df0d86c9df096eefb5fd99a703c479957fc8, not stripped
```

Código 4.7. Comando file em um arquivo ELF com ligação dinâmica

Identificar o tipo de ligação de um binário desconhecido é essencial para aumentar o entendimento sobre o funcionamento deste, uma vez que o tipo de ligação também influencia na realização dos procedimentos de análise. Um efeito colateral da ligação estática é que símbolos anteriormente exportados para auxiliar na resolução dos endereços das funções contidas nas bibliotecas agora são omitidos por não serem mais necessários, o que reduz a quantidade de informação obtida através dos procedimentos de análise estática. Este fato é frequentemente explorado por atacantes de forma a dificultar a engenharia reversa de suas aplicações maliciosas. Por exemplo, caso o mesmo arquivo ELF anteriormente apresentado fosse ligado estaticamente a alguma biblioteca, a saída do comando *file* seria como mostrado no Código 4.8, no qual a linha 3 exibe o tipo de ligação do arquivo e nenhuma informação sobre bibliotecas ou interpretadores é provida. Note ainda que, embora tendo sido compilado a partir do mesmo arquivo fonte, o *hash* do binário compilado (linha 4) difere do binário anterior. A recompilação de arquivos é uma estratégia frequentemente utilizada por atacantes para gerar `variantes` de arquivos maliciosos quando o binário original passa a ser detectado por soluções Antivírus (AVs).

```
1 >$ file static-malicious.bin
2 static-malicious.bin: ELF 64-bit LSB executable, x86-64, version 1 (
     GNU/Linux),
3 statically linked, for GNU/Linux 2.6.32,
4 BuildID[sha1]=a20285090944c95cc21aac376a87144b37657496, not stripped
```

Código 4.8. Comando file em arquivo ELF compilado com ligação estática

4.3.2. O utilitário e comando `LDD`

Quando da identificação de um binário utilizando-se ligação dinâmica, o analista deve proceder a identificação das bibliotecas ligadas de modo a melhor compreender o funcionamento do binário e, eventualmente, prever seu comportamento através dos tipos de bibliotecas ligadas (e.g., bibliotecas de rede sugerem que o *malware* possa ser um *downloader* [Grégio et al. 2015] ou que exfiltre informações sensíveis). Para tanto, pode-se utilizar o comando *List Dynamic Depedencies* (*ldd*), como exemplificado pelo Código 4.9.

```
1 >$ ldd dynamic-malicious.bin
2   linux-vdso.so.1 =>  (0x00007ffe987e6000)
3   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f77a58b3000)
4   /lib64/ld-linux-x86-64.so.2 (0x00007f77a5c7d000)
```

Código 4.9. Comando ldd em arquivo ELF com ligação dinâmica

Caso o arquivo analisado tenha sido ligado estaticamente, por exemplo, por rotinas de ofuscação, o comando *ldd* informará ao analista não ser capaz de identificar nenhuma biblioteca, como exemplificado pelo Código 4.10.

```
1 >$ ldd static-malicious.bin
2   not a dynamic executable
```

Código 4.10. Comando ldd em arquivo ELF com ligação estática

4.3.3. O utilitário e comando `objdump`

Quando um binário apresenta ligação dinâmica, mais do que se identificar a quais bibliotecas este se liga, pode-se identificar quais funções presentes nestas bibliotecas são referenciadas pelo binário. Através da opção `-T`, o comando `objdump` permite analisar a tabela de símbolos dinâmicos dos binários fornecidos, incluindo os símbolos usados pelo ligador dinâmico, tal qual exemplificado pelo Código 4.11.

```
>$ objdump -T VirusShare_4e0ee9f1571107a015e63925626b562d

VirusShare_4e0ee9f1571107a015e63925626b562d:     file format elf32-
    i386

DYNAMIC SYMBOL TABLE:
080484d8      DF *UND*  0000003b  GLIBC_2.0   inet_addr
080484f8      DF *UND*  000000d8  GLIBC_2.0   __libc_start_main
08048508      DF *UND*  00000039  GLIBC_2.0   printf
08048518      DF *UND*  00000046  GLIBC_2.0   memcpy
08048528      DF *UND*  000001c8  GLIBC_2.0   gethostbyname
08048538      DF *UND*  000000e2  GLIBC_2.0   exit
08048548      DF *UND*  0000003e  GLIBC_2.0   atoi
08048558      DF *UND*  00000039  GLIBC_2.0   send
08048568      DF *UND*  0000000e  GLIBC_2.0   htons
08048578      DF *UND*  00000054  GLIBC_2.0   memset
08048588      DF *UND*  00000039  GLIBC_2.0   connect
08048598      DF *UND*  00000039  GLIBC_2.0   recv
080485a8      DF *UND*  00000034  GLIBC_2.0   sprintf
080485b8      DF *UND*  00000039  GLIBC_2.0   socket
```

Código 4.11. Saída do comando objdump exibindo a tabela de símbolos de um exemplar de malware indicando o uso de recursos de rede

A identificação das funções referenciadas pelo binário possibilita que o analista tenha uma primeira ideia dos possíveis comportamentos que o binário possa apresentar. No exemplo acima, identifica-se que este pode fazer significante uso de recursos de rede, dado as referências as funções `connect`, `send`, `recv` e `socket`, utilizadas para implementar diferentes tipos de comunicação entre diferentes máquinas (*hosts*) e/ou à Internet [Yocom et al. 2004].

Cientes da possibilidade de se identificar o comportamento do binário pelos símbolos das funções referenciadas, atacantes empregam diversas técnicas para confundir o analista, como: (i) referenciar muitas funções que não são executadas de fato, apenas para dificultar procedimentos de análise; (ii) ofuscar o código para que símbolos de bibliotecas sejam resolvidos apenas em tempo de execução; ou (iii) compilar bibliotecas e funções estaticamente, escondendo seus símbolos, como exemplificado pelo Código 4.12, já que a tabela de símbolos não precisa ser populada por bibliotecas ligadas estaticamente.

```
>$ objdump -T static-malicious.bin
static-malicious.bin:     file format elf64-x86-64
objdump: static-malicious.bin: not a dynamic object
DYNAMIC SYMBOL TABLE:
no symbols
```

Código 4.12. Saída do comando objdump indicando a ausência da tabela de símbolos para um binário ELF ligado estaticamente

Quando símbolos não estão presentes, pode-se tentar identificar construções maliciosas olhando diretamente para as instruções presentes no binário. Através da opção `-d`, o comando `objdump` permite realizar o *disassembly* da seção de instrução do binário, tal qual exemplificado pelo Código 4.13, no qual pode-se observar uma construção típica de binários que fazem uso de rede: (i) a tentativa de obtenção de um nome de rede (`gethostbyname`); (ii) a verificação do valor de retorno (`cmpl`); resultando no termino da execução (`exit`) em caso de erro (`perror`); ou (iv) na criação de um `socket` caso esta tenha sido bem sucedida.

```
1  >$ objdump -d VirusShare_4e0ee9f1571107a015e63925626b562d
2
3  0804876f <main>:
4  8048ced:       e8 36 f8 ff ff          call   8048528 <
       gethostbyname@plt>
5  8048cf2:       83 c4 10                add    $0x10,%esp
6  8048cf5:       89 85 e4 75 ff ff       mov    %eax,-0x8a1c(%ebp)
7  8048cfb:       83 bd e4 75 ff ff 00    cmpl   $0x0,-0x8a1c(%ebp)
8  8048d02:       75 1a                   jne    8048d1e <main+0x5af>
9  8048d04:       83 ec 0c                sub    $0xc,%esp
10 8048d0c:       e8 a7 f7 ff ff          call   80484b8 <perror@plt>
11 8048d11:       83 c4 10                add    $0x10,%esp
12 8048d14:       83 ec 0c                sub    $0xc,%esp
13 8048d19:       e8 1a f8 ff ff          call   8048538 <exit@plt>
14 8048d1e:       83 ec 04                sub    $0x4,%esp
15 8048d27:       e8 8c f8 ff ff          call   80485b8 <socket@plt>
```

Código 4.13. Saída do comando objdump ilustrando o dissassembly de um exemplar malicioso que faz uso de recursos de rede

Novamente, atacantes estão cientes da possibilidade de se identificar construções maliciosas através do *disassembly* das instruções do binário e empregam técnicas de anti-análise para dificultar a tarefa do analista. Entre as técnicas, destacam-se: (i) ofuscação do binário, fazendo com que as instruções maliciosas não sejam diretamente visíveis ao *disassembler*; (ii) técnicas de anti-*disassembly* [Branco et al. 2012], fazendo com o que o *disassembler* produza resultados incorretos; e (iii) a inserção de código morto (*dead-code*), código *assembly* sem qualquer utilidade real apenas para dificultar a tarefa de análise. Para superar estas limitações, o analista deve proceder para a análise dinâmica, a fim de observar o código efetivamente executado pelo binário.

4.3.4. O utilitário e comando `strings`

Além de tabelas de símbolos, a estrutura dos binários ELF carrega outras informações que podem ajudar a inferir o comportamento dos binários desconhecidos, como *strings* estáticas utilizadas pelo programador. *Strings* são alocadas na seção de dados estáticos do binário, cujos *bytes* podem ser interpretados como caracteres `ASCII`, como exemplificado na Figura 4.2.

As mesmas informações sobre *strings* contidas no exemplar podem ser obtidas através do comando *strings*, exibido no Código 4.14, que apresenta uma exibição mais facilitada para a interpretação por seres humanos. Pode-se notar a presença de diversas *strings* representando endereços IP, o que sugere a possibilidade deste exemplar fazer significante número de comunicações via rede. Em particular, o uso de formatadores de

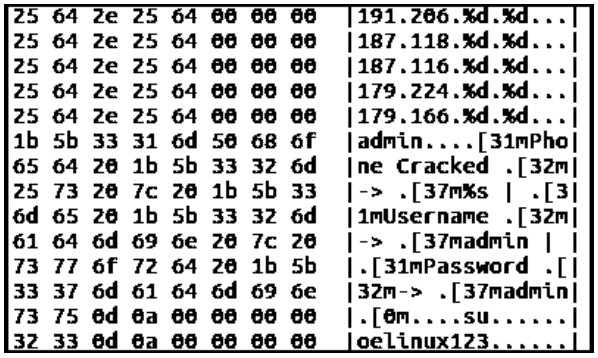


Figura 4.2. Comando hexdump -C em exemplar malicioso contendo *strings* suspeitas.

strings do tipo inteiro (`%d`), sugere que este exemplar pode ser um *scanner* [Galante et al. 2018, Cozzi et al. 2018] de rede, realizando tentativas contra todos os *hosts* possíveis dentro da sub-rede especificada. Tal hipótese é corroborada pelas demais *strings*, que ilustram diversas senhas que podem ser utilizadas em ataques de força bruta [Wang et al. 2018].

```
>$ strings
   a9a780c66ec18861e4881430202f62d6ceaba3187626d48ab241522f86a5c254
189.96.%d.%d            189.99.%d.%d
39.34.%d.%d             59.103.%d.%d
191.12.%d.%d            186.117.%d.%d
179.170.%d.%d           191.206.%d.%d
187.118.%d.%d           179.224.%d.%d
admin                   [31mPhone Cracked
[31mUsername            [31mPassword
[36mDevice Cracked      GETLOCALIP
My IP: %s               BOTKILL
Killing Bots            NETIS ON | OFF
[TELNET] SCANNER ON:%s  ICMP
HTTP                    STOP
ENDTHEBOTLOL            8.8.8.8
/proc/net/route         /etc/resolv.conf
```

Código 4.14. Saída do comando strings em exemplar malicioso indicando diversas strings suspeitas

Como nos casos anteriores, atacantes previnem-se de exibir suas *strings* através de: (i) geração de *strings* em tempo de execução; e (ii) ofuscação das *strings* estáticas, como exemplificado pelo Código 4.15.

```
>$ strings upx-file.bin
$Info: This file is packed with the UPX executable packer http://
   upx.sf.net $
dl%)
UPPH
q}Nsf
x`TJq&H
UPX!
```

Código 4.15. Saída do comando strings em arquivo empacotado pela solução UPX

4.3.5. O utilitário e comando `UPX`

Uma das formas mais comuns de se ofuscar códigos maliciosos é através do uso de empacotadores, capazes de embutir o arquivo original em sua estrutura e extraí-los em tempo de execução e dentre os quais a solução *Ultimate Packer for eXecutables* (*UPX*) [M.F.X.J. Oberhumer 2018] é a mais popular. O Código 4.16 ilustra o empacotamento de um arquivo com ligação estática via *UPX*.

```
>$ upx original.bin -o upx-file.bin
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep
    30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     610309 ->    270256   44.28%  linux/elf386   1-upx

Packed 1 file.
```

Código 4.16. Empacotamento de um binário com ligação estática pela solução UPX

O binário empacotado não somente ofusca o conteúdo do binário malicioso original, mas também pode confundir soluções antivírus (AVs), já que estas frequentemente comparam o *hash* do arquivos suspeitos contra bases de *hashes* de binários conhecidos [Koret and Bachaalany 2015] e a compressão por `UPX` resulta em um binário com *hash* distinto do original, como exemplificado pelo Código 4.17. Este tipo de estratégia é deliberadamente utilizada por atacantes não apenas utilizando-se de compressores como `UPX`, mas diversas ferramentas, notadamente *crypters* [Tasiopoulos and Katsikas 2014].

```
b9c93c4cf9f0848f03834614c6f91759e4e068c0  original.bin
3a5d19d3372f4d8e05599da542bfa161a14a4a32  upx-file.bin
```

Código 4.17. Diferença no sha1sum dos binários original e empacotado

Felizmente, a ferramenta *UPX* permite desempacotar o arquivo empacotado como exibido no Código 4.18, de modo a retornar o arquivo ao estado original, permitido, assim, a análise do exemplar não ofuscado. Contudo, este é um caso particular, visto que muitos empacotadores requerem procedimentos complexos para que os binários gerados por estes possam ser desempacotados [Cheng et al. 2018, Coogan et al. 2009].

```
>$ upx -d upx-file.bin
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep
    30th 2013
        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     610311 <-    270256   44.28%  netbsd/elf386  1-upx
Unpacked 1 file.
```

Código 4.18. Desempacotamento de um binário empacotado via UPX

4.4. Modificações em Arquivos Binários

Procedimentos de análise estática, como mostrado na Seção 4.3, são úteis para se obter informações dos binários de modo passivo, isto é, sem interferir na estrutura do binário em questão. Apesar de efetivo em muitos casos, muitas construções maliciosas, como técnicas de ofuscação, podem impedir a obtenção de dados significativos para a caracterização do comportamento do binário como malicioso ou benigno. Nestes casos, o analista pode optar por alterar a estrutura do binário de modo a exibir informações ocultas. Pode-se, por exemplo, contornar as rotinas de proteção e/ou ofuscação de modo a se inspecionar o real comportamento do binário. Apresenta-se, a seguir, uma estratégia de modificação de binários (*binary patching*) útil para estes casos.

4.4.1. *Binary Patching*

A técnica de *binary patching* consiste em alterar diretamente os *bytes* do binário sob análise de modo que este revele algum comportamento de interesse do analista. Embora a alteração possa ocorrer em qualquer seção do binário, esta é usualmente empregada sob as instruções de modo a contornar alguma rotina de verificação implementada pelo atacante. Por exemplo, o exemplar ilustrado pelo Código 4.19 requer que uma senha seja digitada para que a execução continue. Caso o analista não seja capaz de descobrir a senha por outros modos, a eliminação desta rotina de verificação é o único modo de se observar o progresso da execução deste binário.

```
1 int password()
2   scanf("%[^\n]s",string);
3   if(strcmp(string,PASS)==0)
4     malicious();
5   else
6     exit(0);
```

Código 4.19. Exemplo de código de um exemplar que exibe seu comportamento malicioso apenas quando a string de inicialização é corretamente definida.

```
400692      call        wrapper_601030_400520
400697      test        eax, eax
400699      jnz         loc_4006a5
40069b      mov         edi, strz_Oh_Yeah__40075a
4006a0      call        wrapper_601018_4004f0
4006a5
```

Figura 4.3. *Disassembly* da função verificadora. A função verificadora é invocada (linha 1) e seu valor de retorno é testado (linha 2) de modo a determinar se a execução deve proceder (linha 4) ou não (linha 3).

Técnicas de *binary patching* são independentes de ferramentas, já que são diretamente aplicadas ao binário alvo. Contudo, para fins didáticos, nos referiremos aqui a solução *HT Editor* (*hte*) [S. Weyergraf 2015]. Navegando pelo *hte* através de `F6 ->`

`elf/image` e `F8 -> main`, obtém-se a visão (*view*) ilustrada pela Figura 4.3. Nesta, identifica-se a sequência de instruções que implementa o comportamento exibido pelo Código 4.3: *call* para chamar a função *strcmp* comparar as *strings*; *test* para realizar a comparação do valor de retorno; e *jnz* para mudança no fluxo de execução caso o valor de retorno não seja o esperado, ou seja, senha incorreta.

Uma das formas de se forçar o prosseguimento da execução mesmo sem o conhecimento da senha requerida é eliminar a instrução *jnz*, de modo que o programa sempre execute o trecho seguinte. Na prática, contudo, não é possível meramente eliminar uma instrução, pois isto corromperia a estrutura do binário e desalinharia as regiões de memória. Desta forma, uma alternativa viável é substituir a instrução por uma de tamanho equivalente mas que não execute nenhuma ação concreta, como, por exemplo, a instrução *nop* (*no operation*). Para o exemplo apresentado, como a instrução *jnz* ocupa 2 *bytes* de memória, duas instruções *nop* de 1 byte cada são requeridas. Esta substituição pode ser realizada no *hte* via navegação (*F4*) até a instrução *jnz* e substituir pelo código `0x90` (`nop` duas vezes, como exibido na Figura 4.4. O binário alterado pode ser salvo (*F2*) e executado normalmente.

```
400692      call        wrapper_601030_400520
400697      test        eax, eax
400699      nop
40069a      nop
40069b      mov         edi, strz_Oh_Yeah__40075a
4006a0      call        wrapper_601018_4004f0
```

Figura 4.4. *Disassembly* da função verificadora. A função verificadora pode ser modificada de modo que a rotina de verificação seja eliminada.

4.4.2. Verificações de integridade

Assim como atacantes estão cientes de que seus códigos serão inspecionados por procedimentos passivos de análise estática, estes também estão cientes de que analistas tentarão modificar os binários distribuídos. Como contramedida, atacantes implementam rotinas de verificação de integridade, de modo a impedir ou dificultar a modificação das funções de verificação implementadas nos binários maliciosos. Um tipo popular de verificação é o calculo do *Cyclic Redundancy Check* (*CRC*) [Peterson and Brown 1961]. O Código 4.20 ilustra uma versão modificada do exemplo anterior na qual a função *passwd*, responsável por verificar a senha, possui um *CRC* calculado previamente (*MYCRC*), o qual também é verificado durante a execução, de modo que uma alteração na função devido a *binary patching* resulte no encerramento da aplicação.

```
1 int main()
2     crc32 = Crc32_ComputeBuf(0,
          passwd,33);
3     if(crc32!=MYCRC)
4         exit(0);
5     passwd(pass);
```

Código 4.20. Exemplo de código de verificação de CRC32 para proteger a função verificadora de strings.

Para realizar análise do exemplar com verificação de integridade via *CRC*, deve-se, agora, realizar *binary patching* em duas seções do exemplar: (i) na função *passwd* para ignorar o resultado da verificação da senha, tal qual no caso anterior; e (ii) na rotina de verificação de integridade da função *passwd* via `CRC`. Através do uso do *hte*, pode-se realizar *binary patching* das duas seções; a primeira tal qual demonstrado anteriormente, e a segunda tal qual exibido na Figura 4.5, onde a instrução *jz* foi substituída pela instrução *jnz* de modo a se evitar o término da execução mesmo com a falha de verificação de integridade da função modificada. Resultado semelhante pode ser obtido de forma dinâmica, como mostrado na Seção 4.6.

```
4007fd      call        Crc32_ComputeBuf
400802      mov         [rbp-    ], rax
400809      cmp         qword ptr [rbp-    ],
400814      jnz         loc_400820
400816      mov         edi,
40081b      call        wrapper_602050_400680
```

Figura 4.5. *Disassembly* da região ao redor da função verificadora.. Rotina de verificação do CRC32 é invocada (linha 1) e resulta no término da execução (linha 4) caso a função original tenha sido alterada.

4.5. Traços de Execução

A análise estática permite ao analista obter informações preliminares sobre o possível comportamento do binário suspeito e, assim, traçar um plano de investigação concreto. Contudo, devido as limitações discutidas na Seção 4.3, a confirmação das suspeitas se dá apenas através do emprego de métodos de análise dinâmica, com a execução do artefato suspeito em um ambiente controlado, denominado *sandbox* [Sikorski and Honig 2012]. Este ambiente deve limitar a ação do *malware* de modo a evitar que uma infecção se propague. No âmbito deste curso, recomenda-se executar os experimentos em uma máquina virtual desconectada da Internet.

O produto mais comum da execução de um binário em uma *sandbox* é um traço de execução: uma lista sequencial de ações realizadas pelo binário. As ações podem incluir chamadas de sistema (*syscalls*), de aplicações (API), de instruções de máquina executadas, entre outros. Nesta seção, descrevemos soluções para obtenção dos três tipos de traço de execução anteriormente referidos.

Contudo, assim como atacantes implementam técnicas de ofuscação para dificultar procedimentos de análise estática, atacantes também implementam técnicas de anti-análise para evadir a execução em *sandboxes*. Apesar de cumprir papel semelhante, a estratégia de evasão se difere da de ofuscação pois enquanto a última apenas mascara os dados, a segunda tenta impedir o procedimento de análise em si. A seguir, detalha-se estratégias para a obtenção de traços de execução de aplicações e para o contorno de técnicas de evasão de análise.

É importante notar que a plataforma Linux possui diversas soluções para *tracing* de binários (uma lista completa pode ser encontrada no *survey* [Gebai and Dagenais

2018]) e que, por questões didáticas, limitamo-nos a apresentar exemplos de utilização das soluções mais populares e que detalhes de todas as chamadas de sistemas apresentadas nos traços de execução podem ser consultados via *manpages* [Kernel.org 2017].

4.5.1. A ferramenta e o comando `strace`

Uma das soluções mais populares para a obtenção de traços de execução a nível de chamadas de sistema (*syscall*) é o *strace* (*system call tracer*), que reporta, sequencialmente, as chamadas de sistemas executadas em conjunto dos argumentos passados para estas e dos valores de retorno destas.

O Código 4.21 ilustra um processo realizando a chamada de sistema *open*, com `strace` reportando o *pid* do processo (`11047`), os parâmetros da chamada (`/etc/passwd`) e o valor de retorno (`-1`). Neste exemplo, realizado por um exemplar malicioso, o processo tenta acessar o arquivo `passwd`, que contém informação das senhas de usuários, um tipo de ação característica de exemplares maliciosos que tentam realizar roubo de credenciais. Para este caso específico, o valor de retorno `-1` indica que o acesso a este arquivo foi negado, o que era esperado dado que, em sistemas típicos, o arquivo `passwd` só pode ser acessado com permissões de super-usuário (*root*).

```
1 [pid 11047] open("/etc/passwd", O_WRONLY|O_CREAT|O_APPEND, 0666) = -1
      EACCES (Permission denied)
```

Código 4.21. Exemplo de registro de chamada de sistema via strace de uma tentativa de acesso a credenciais do usuário (/etc/shadow).

Enquanto uteis para identificar padrões de acesso e comportamentos maliciosos (uma lista detalhada de comportamentos maliciosos é apresentada no artigo [Grégio et al. 2012]), traços de execução, por conterem todas as chamadas que um binário executa, apresentam muitas informações que não são essenciais aos procedimentos de análise, como rotinas de inicialização e de *clean up*, causando poluição dos arquivos de *log*. O Código 4.22 ilustra um exemplar malicioso sendo inicializado (carregando bibliotecas via `access`) e, posteriormente, já próximo ao termino de sua execução, exibindo seu código de *usage*, no caso, requisitando os endereços IP e porta para executar corretamente, visto que este exemplar realiza ataques via rede. Observa-se, ao final do traço, a chamada de sistema para o término do processo: `exit_group` com valor menos um, dado que a execução esperada (com parâmetros corretos) falhou.

```
1  >$ strace ./malware.bin
2  execve("./malware.bin", ["./malware.bin"], [/* 25 vars */]) = 0
3  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
4  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT
5  ...
6  write(2, "Invalid␣parameters!\n", 20Invalid parameters!)   = 20
7  write(1, "DNS␣Flooder␣v1.3\nUsage:␣./malwar"..., 244DNS Flooder v1.3
8  Usage: ./malware.bin <target IP/hostname> <port to hit> <reflection
       file (format: IP DOMAIN>> <number threads to use> <time (optional
       )>
9  exit_group(-1)                          = ?
10 +++ exited with 255 +++
```

Código 4.22. Traço de chamadas de sistema de exemplar malicioso exibindo rotinas de carregamento e de finalização

Uma vantagem significativa do `strace` sobre outros métodos de análise é que os traços de execução obtidos por este independem do tipo de ligação (estática ou dinâmica) entre binários e bibliotecas. Desta forma, ainda que o atacante tenha conseguido esconder as funções referenciadas em tempos de compilação, estas aparecerão nos traços dinâmicos caso sejam invocadas. Portanto, a única observação divergente entre traços de execução de binários estáticos e dinâmicos é a presença de diversas funções de carregamento de bibliotecas executadas pelas rotinas de inicialização da aplicação, tal qual ilustrado pelo Código 4.23.

```
1  strace 0
       c4f1bf21f943331f9952a594fa37a868708a9458180c2a156a08ae9a9a1d29f
2  execve("./malware.bin", ["./malware.bin"], [/* 25 vars */]) = 0
3  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
       directory)
4  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
       directory)
5  open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
6  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
       directory)
7  open("/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
8  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
       directory)
9  open("/usr/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC)
       = 3
10 open("/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
11 open("/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
12 open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
13 rt_sigaction(SIGINT, {0x408de1, [INT], SA_RESTORER|SA_RESTART, 0
       x7fbc2ac834b0}, {SIG_DFL, [], 0}, 8) = 0
14 rt_sigaction(SIGILL, {0x408de1, [ILL], SA_RESTORER|SA_RESTART, 0
       x7fbc2ac834b0}, {SIG_DFL, [], 0}, 8) = 0
```

Código 4.23. Traço de chamadas de sistema de binário carregando diversas bibliotecas

Embora o *strace* não apresente restrições quanto a análise de binários compilados estaticamente, atacantes podem dificultar o trabalho do analista ao dividir as ações maliciosas em múltiplos processos. Nestes casos, caso o analista obtenha um traço apenas do processo inicial, não observará os comportamentos associados aos demais processos. Contudo, ao "seguir" os processos filhos criados pelo processo inicial, os comportamentos podem ser observados normalmente.

O Código 4.24 ilustra o funcionamento de um exemplar malicioso que realiza a chamada de sistema *clone* (equivalente a *fork*), que cria um processo filho, idêntico ao processo pai, mas com identificador (*pid*) diferente. O processo filho começa sua execução do ponto de chamada do processo pai e com os mesmos valores de variáveis. Contudo, por ser um processo independente, ações diferentes do processo pai podem ser executadas com o prosseguimento da execução. A análise do traço de exemplo permite identificar tais diferenças, pois, após realizar *clone*, o processo pai entra em espera *wait* para que o filho assuma a execução. O filho, que carrega o conteúdo malicioso (*payload*), por sua vez, evade a análise por detectar o ambiente *ptrace*, como será discutido em detalhes no futuro. O rastreio de ações do processo filho pode ser executado inicializando a ferramenta *strace* com o argumento `-f`.

```
1  >$ strace -f
       d73a9a8dbd1a4a73048d973457eaa854fe19c689bcd8f87395d65ae8b1d7587d
2  execve("./malware.bin", ["./malware.bin"], [/* 25 vars */]) = 0
3  brk(NULL)                                = 0x240b000
4  access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT (No such file or
       directory)
5  access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (No such file or
       directory)
6  ...
7  clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
       SIGCHLD, child_tidptr=0x7fbc2bce09d0) = 11045
8  wait4(-1, strace: Process 11045 attached
9   <unfinished ...>
10 [pid 11045] ptrace(PTRACE_TRACEME, 0, NULL, NULL) = -1 EPERM (
       Operation not permitted)
11 [pid 11045] execve("./malware.bin", ["./malware.bin", "2", "3", "4"],
        [/* 25 vars */]) = 0
12 ...
```

Código 4.24. Traço de chamadas de sistema de exemplar malicioso com chamada fork e evasão de execução por parte do processo filho

Outra técnica de evasão de análise que também afeta o *strace* é o *delay* na execução de chamadas, comumente implementado através de chamada da função *sleep* por um tempo suficientemente longo para que a análise termine antes dos comportamentos maliciosos serem exibidos [Grégio et al. 2012], seja durante uma sessão de depuração ou durante a execução em um ambiente de análise automatizado limitado por um *timeout*. O Código 4.25 ilustra a ocorrência deste tipo de técnica de evasão durante um traço de execução de um processo malicioso. Nota-se que nenhuma outra chamada de sistema é reportada após a invocação da função *nanosleep* dado que esta não retornou dentro do período de análise deste binário. Um método para mitigar os efeitos deste tipo de técnica de evasão é discutido na Seção 4.6.2.

```
1 >$ strace
      a4332ab4b8f8e2f04fef7c40c103ab570f42011ba41b3caaa03029b8928cba2e
2 [pid 11046] waitpid(11084, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}],
      0) = 11084
3 [pid 11046] munmap(0xf6f0a000, 4096)    = 0
4 [pid 11046] ioctl(1, SIOCGIFFLAGS, {ifr_name="ens3", ifr_flags=IFF_UP
      |IFF_BROADCAST|IFF_RUNNING|IFF_MULTICAST}) = 0
5 [pid 11046] gettimeofday({312344432895491, -17819621742608320}, NULL)
       = 0
6 [pid 11046] nanosleep({429496729600000000, 577756380723720712},  <
      unfinished ...>
```

Código 4.25. Traço de chamadas de sistema de exemplar malicioso que realiza uma chamada sleep com tempo suficientemente longo para causar o término da execução em muitos sistemas de análise

Finalmente, uma outra forma de evadir análise baseadas em `strace` é explorar os efeitos colaterais das decisões de projeto para a construção da própria ferramenta `strace`, que é baseada no *framework* `ptrace` [Kernel.org 2017], uma poderosa solução de monitoração nativa da plataforma `Linux` (implementações baseadas em `ptrace` são

discutidas na Seção 4.5.3). `Ptrace` estabelece um *lock* quando anexado a um processo, de modo que nenhum outro processo possa se anexar ao mesmo processo sendo analisado. Naturalmente, ao se analisar um binário usando `strace`, este *lock* é estabelecido pela solução de *tracing*.

O estabelecimento deste *lock* implica na possibilidade de detecção do ambiente `ptrace` por parte de binários maliciosos casos estes possam detectar que o *lock* tenha sido estabelecido para o seu próprio processo malicioso. Atacantes podem detectar o *lock* ao tentar estabelecer uma ligação `ptrace` consigo mesmo, pois esta falhará caso o *lock* já tenha sido estabelecido por algum processo anterior (no caso, o `strace`). Deste modo, atacantes podem evadir processos de análise quando da identificação da execução em uma solução de *tracing*. Uma possível implementação deste tipo de evasão é ilustrado no Código 4.26

```
1 int main(){
2   if(ptrace(PTRACE_TRACEME)==-1)
3      exit(0);
4   malicious()
```

Código 4.26. Evasão de análise por strace através da autoverificação da presença do framework ptrace.

Caso este código seja executado em `strace`, a análise será encerrada, como exemplificado pelo Código 4.27. Contudo, sua execução em ambientes nativos, como as máquinas das vítimas/usuários, o código executa naturalmente e exibe seus comportamentos maliciosos. Este tipo de técnica de evasão pode ser contornada através da modificação do fluxo de execução, como descrito na Seção 4.6.

```
1 >$ strace ./ptrace
2 ...(ommitted initialization system calls)
3 ptrace(PTRACE_TRACEME, 18446744073049584056, 0
     x7ffdd8a9b1c8, NULL) = -1 EPERM (Operation
     not permitted)
4 exit_group(0)                           = ?
5 +++ exited with 0 +++
```

Código 4.27. Traço de chamadas de sistema de arquivo com evasão de análise via chamada ptrace

4.5.2. Ltrace

A solução de análise de traço de execução *ltrace* é o análogo da solução *strace* para a obtenção de traços de execução a nível de chamadas de funções (APIs). O traço a nível de API é útil para determinar comportamentos maliciosos em mais alto nível, uma vez que diferentes APIs (`printf, putc`) podem resultar na mesma chamada de sistema (`write`). O Código 4.28 exemplifica o traço de um exemplar malicioso que carrega o *exploit DirtyCow* [Oster 2019] para escalar privilégios e copiar o arquivo de senhas *passwd*.

```
1 >$ ltrace
     b0148c40166b2b6cea71e1f00100a05087d2a492f21e008e30c594cf91ecf9f1.ltr
2 [pid 11043] __libc_start_main(0x400ab2, 1, 0x7ffc53304f98, 0x400d00 <
     unfinished ...>
3 [pid 11043] printf("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"..., "/usr/bin/
     passwd") = 308
4 [pid 11043] puts("DirtyCow␣root␣privilege␣escalati"...) = 35
5 [pid 11043] printf("Backing␣up␣%s␣to␣/tmp/bak\n", "/usr/bin/passwd")
     = 39
6 [pid 11043] asprintf(0x7ffc53304ea0, 0x400f1e, 0x6020c0, 0x7fffffe5)
     = 27
7 [pid 11043] system("cp␣/usr/bin/passwd␣/tmp/bak" <no return ...>
8 ...
```

Código 4.28. Traço de chamadas de funções de exemplar malicioso executando um exploit para a vulnerabilidade DirtyCow

Tal qual `strace`, análises baseadas em `ltrace` devem "seguir" processos filhos (`fork`) para evitar evasão. Similarmente, `ltrace` também é vulnerável a evasão por longos *delays* e possivelmente por detecção de *locks* do tipo `ptrace`. Além disso, atacantes também exploram outro fator derivado das decisões de projeto para a implementação de `ltrace`: dado que `ltrace` instrumenta as APIs monitoradas através da alteração da tabela de símbolos, este só pode lidar com binários ligados dinamicamente. Em caso de binários estáticos, `ltrace` exibe um erro, como ilustrado no Código 4.29.

```
1 >$ ltrace ./static-malicious.bin
2 Couldn't␣find␣.dynsym␣or␣.dynstr␣in␣"/proc/25332/exe"
```

Código 4.29. Traço de chamadas de funções reportando falha ao analisar um binário estaticamente ligada via ltrace

4.5.3. Ptrace

Soluções como *strace* baseiam seu funcionamento no *framework ptrace*, que fornece uma interface clara para o monitoramento de aplicações. Através da chamada `ptrace`, tal qual mostrada no Código 4.30, um processo (`monitor`) pode monitorar um outro processo (`monitorado`) especificado através de um *Process IDentifier* (PID). Em arquiteturas de monitoramento típicas, tais quais as implementadas por `strace` e `ltrace`, o processo monitor invoca `ptrace` sobre um processo filho, o binário monitorado.

```
1 long ptrace(enum __ptrace_request request, pid_t pid, void *addr,
     void *data);
```

Código 4.30. Protótipo para a chamada ptrace

`Ptrace` é usado como suporte para a construção de diferentes tipos de monitores, de *traces* como os apresentados a *debuggers*, como o GDB [Alves et al. 2017]. Isto é possível devido as diversas capacidades de monitoração providos por este, especificadas pela *enum ptrace_request*, e que incluem: obtenção dos valores nos registrador do processador (`PTRACE_GETREGS`), de valores e argumentos de chamadas de sistema (`PTRACE_SYSCALL`), e até mesmo execução *single-step* (`PTRACE_SINGLESTEP`).

Consulte o repositório indicado na introdução deste capítulo para a obtenção de exemplos de como programar usando `ptrace`.

Considerando as capacidades de monitoração providas por `ptrace` e as capacidades de desenvolvimento da plataforma `Linux`, analistas podem desenvolver suas próprias soluções de análise, customizando-as de acordo com suas necessidades. Uma solução de análise interessante de ser desenvolvida a partir de `ptrace` é um *instruction tracer* (`itracer`), através das capacidades de execução e inspeção *single-step*, o que permite obter detalhes da execução de processos com informações de baixo nível. O Código 4.31 ilustra o traço de execução de um arquivo malicioso quando este realiza mudanças do fluxo de execução entre o binário monitorado (`itrace`) e uma biblioteca ligada (`ld-2.23.so`). Os endereços providos permitem ao analista identificar com exatidão os pontos de troca de contexto.

```
1  >$ itrace
       f5d5557da51fb1ac8da1a84fc9a6783496cadf2be438110995dbebe6ae0337d4
2  RIP: 0x400c63 | /home/SBSeg/itrace 0x400000 0x403000
3  RIP: 0x400c6a | /home/SBSeg/itrace 0x400000 0x403000
4  RIP: 0x7f446f862c17 | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
5  RIP: 0x7f446f862c1a | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
6  RIP: 0x7f446f862c1e | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
7  RIP: 0x7f446f862c20 | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
8  RIP: 0x7f446f862c27 | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
9  RIP: 0x7f446f862c2a | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
10 RIP: 0x7f446f862c2c | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
11 RIP: 0x7f446f862c30 | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
12 RIP: 0x7f446f862c33 | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
13 RIP: 0x401d84 | /home/SBSeg/itrace 0x400000 0x403000
14 RIP: 0x401d88 | /home/SBSeg/itrace 0x400000 0x403000
15 RIP: 0x401d8c | /home/SBSeg/itrace 0x400000 0x403000
16 RIP: 0x7f446f862c35 | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
17 RIP: 0x7f446f862c38 | /lib/x86_64-linux-gnu/ld-2.23.so 0x7f446f852000
       0x7f446f878000
```

Código 4.31. Traço de execução de um exemplar malicioso a nível de instruções. Pode-se identificar os pontos exatos de troca de contexto entre a execução do código do binário e da biblioteca ligada.

4.6. Alteração de contexto

Tal qual abordagens estáticas podem ser caracterizadas como passivas (Seção 4.3) ou ativas (Seção 4.4), abordagens dinâmicas também podem envolver apenas a observação da execução (Seção 4.5) ou a modificação ativa dos fluxos de execução, como apresentado nesta seção. Técnicas de modificação de fluxo consistem em alterar o caminho de ex-

ecução natural da aplicação, fazendo, por exemplo, que um salto (*branch*) seja tomado mesmo quando as condições lógicas para tal não são satisfeitas ou invocando uma função fora do seu contexto. Este tipo de técnica é útil para contornar rotinas de anti-análise (escapando do seu salto) e descobrir rotinas ocultas (invocando funções não referenciadas naquele contexto). Apresentamos, a seguir, duas estratégias para a modificação de contexto com foco na análise de binários maliciosos.

4.6.1. *Debugging*

Uma das soluções mais populares para a depuração (*debugging*) de código em plataformas Linux é o GDB [Alves et al. 2017], que também pode ser usado para realizar a engenharia reversa de aplicações, tal qual exemplificado nesta seção. O nosso objetivo não é esgotar todas as possibilidades de uso da ferramenta, mas apresentar algumas possibilidades importantes ao leitor, de modo que este possa, futuramente, se aprofundar no assunto através da literatura especializada em GDB [Matloff and Salzman 2008] ou engenharia reversa [Wong 2018].

Uma das capacidades do GDB é realizar o *disassembly* do código binário, como exemplificado pelo Código 4.32. Embora a solução `objdump` mostrada na Seção 4.3 também seja capaz de realizar esta tarefa, o código *assembly* exibido pelo GDB se baseia na interpretação dos *bytes* do binário carregados na memória e está sujeito as modificações de contexto realizadas pelo analista.

```
1  (gdb) disassemble main
2  Dump of assembler code for function main:
3     0x0000000000400646 <+0>:     push   %rbp
4     0x0000000000400647 <+1>:     mov    %rsp,%rbp
5  => 0x000000000040064a <+4>:     sub    $0x110,%rsp
6     0x0000000000400651 <+11>:    mov    %fs:0x28,%rax
7     0x000000000040065a <+20>:    mov    %rax,-0x8(%rbp)
8     0x000000000040065e <+24>:    xor    %eax,%eax
9     0x0000000000400660 <+26>:    mov    $0x400744,%edi
10    0x0000000000400665 <+31>:    callq  0x4004f0 <
          puts@plt>
```

Código 4.32. Disassembly realizado pelo GDB.

Por se tratar de uma solução de inspeção dinâmica, o GDB permite que o analista suspenda a execução em determinados pontos de interesse, denominados *breakpoints* ou pontos de parada, como exemplificado pelo Código 4.33. Este recurso permite que o analista se foque nos dados ao redor da região de interesse e inspecione o programa no exato momento em que uma atividade suspeita está sendo executada pelo binário desconhecido.

```
1  (gdb) b main
2  Ponto de parada 1 at 0x40064a
3  (gdb) r
4  Starting program: /home/sbseg/malware
5  Breakpoint 1, 0x000000000040064a in main ()
```

Código 4.33. Definição de Breakpoints no GDB.

Uma das principais informações a serem obtidas quando um ponto de parada é atingido é a verificação dos valores armazenados nos registradores, como exibido no Código 4.34. Os valores armazenados nos registradores indicam o estado da aplicação, como os registradores de propósito geral (`rax-rdx` e `r8-r15`), *flags* de processamento que indicam se alguma operação resultou em zero, *overflow*, entre outros, e o endereço da instrução sendo executado (`rip`). Note que, neste caso, o endereço apontado por `rip` é o endereço do ponto de parada.

```
(gdb) info registers
rax 0x400646            rbx    0x0
rcx 0x0                 rdx    0x7fffffffdd78
rsi 0x7fffffffdd68      rdi    0x1
rbp 0x7fffffffdc80      rsp    0x7fffffffdc80
r8  0x400730            r9     0x7ffff7de7ac0
r10 0x846               r11    0x7ffff7a2d740
r12 0x400550            r13    0x7fffffffdd60
r14 0x0                 r15    0x0
rip 0x40064a <main+4>   eflags 0x246 [ PF ZF IF ]
```

Código 4.34. Obtenção dos valores atuais nos registradores do programa em execução através do GDB.

Os locais de análise não precisam ser limitado pelos pontos de parada, de modo que um analista pode avançar a execução de forma granular a partir destes para melhor inspecionar uma região de interesse. Uma forma típica de se inspecionar uma aplicação é realizar sua execução passo a passo (*step-by-step*). O Código 4.35 ilustra a execução passo a passo a partir do ponto de parada anteriormente estabelecido através do comando `stepi`, exibindo a instrução executada em cada etapa através da definição `display/i $pc`.

```
(gdb) display/i $pc
Breakpoint 1, 0x000000000040064a in main ()
1: x/i $pc
=> 0x40064a <main+4>:   sub    $0x110,%rsp
(gdb) stepi
0x0000000000400651 in main ()
1: x/i $pc
=> 0x400651 <main+11>:  mov    %fs:0x28,%rax
(gdb)
0x000000000040065a in main ()
1: x/i $pc
=> 0x40065a <main+20>:  mov    %rax,-0x8(%rbp)
(gdb)
0x000000000040065e in main ()
1: x/i $pc
=> 0x40065e <main+24>:  xor    %eax,%eax
```

Código 4.35. Avanço de execução passo a passo.

Fazendo uso dos recursos anteriormente apresentados, analistas podem identificar uma região de interesse, como um ponto de evasão, e modificá-lo de forma a dar

prosseguimento ao processo de análise. Por exemplo, no caso da identificação de uma rotina de evasão implementada através de uma construção do tipo `if-else`, que resulta em instruções de desvio (*branch*) quando interpretadas em `assembly`, o analista pode inverter o caminho originalmente tomado pela aplicação, de modo que não se execute a rotina de evasão mas sim as rotinas maliciosas. Dado que instruções de desvio são controladas pelo registrador de *flags*, a modificação do caminho executado pode se dar pela modificação das *flags* armazenadas neste registrador. O Código 4.36 ilustra um trecho de uma sessão de depuração na qual o registrador de *flags* é alterado, respectivamente, para incluir e remover a flag zero (`ZF`), representada pelo *byte* `0x40`. Consulte a referência [C-Jump 2017] para obter informações sobre a representação de todas as demais *flags*.

```
(gdb) set $eflags|=0x40
(gdb) info registers eflags
eflags         0x246    [ PF ZF IF ]
(gdb) set $eflags|=~0x40
(gdb) info registers eflags
eflags         0x54fd7  [ CF PF AF ZF SF TF IF DF OF NT RF AC ]
```

Código 4.36. Alteração do registrador de flags via GDB.

Além de valores em registrador, a depuração via GDB também permite a alteração de valores em memória, o que permite ao analista fazer uso de diversas técnicas, incluindo *binary patching*, como mostrado na Seção 4.4. O Código 4.37 ilustra um trecho de uma sessão de depuração que realiza o *patch* do binário tal qual mostrado na Seção 4.4. Este tipo de alteração em memória é muito útil para a realização de testes sem a necessidade de se salvar o binário modificado.

```
(gdb) disassemble main
Dump of assembler code for function passwd:
   0x0000000000400692 <+76>:    callq  0x400520 <strcmp@plt>
   0x0000000000400697 <+81>:    test   %eax,%eax
   0x0000000000400699 <+83>:    jne    0x4006a5 <main+95>
(gdb) set {int}0x0000000000400699 = 0x90
(gdb) set {int}0x000000000040069a = 0x90
Dump of assembler code for function passwd:
   0x0000000000400692 <+76>:    callq  0x400520 <strcmp@plt>
   0x0000000000400697 <+81>:    test   %eax,%eax
   0x0000000000400699 <+83>:    nop
   0x000000000040069a <+84>:    nop
```

Código 4.37. Binary Patching via GDB.

As capacidades de alteração de fluxo do GDB não se limitam a alteração dos valores de contexto das aplicações analisadas, mas também pode se dar pela invocação de funções de maneira independente do contexto, de modo que analistas possam descobrir funções ocultas dentro de binários, uma prática muito comum em exemplares maliciosos. O Código 4.38 ilustra a chamada da função verificadora de senha exibida na Seção 4.4 de modo independente das verificações de integridade, o que possibilita a execução da mesma sem a necessidade de se realizar um *patch* do binário. Enquanto o valor de retorno negativo para a primeira chamada indica que a verificação falhou, o valor de retorno positivo para a segunda chamada indica que a senha foi identificada.


```
(gdb) call passwd("MYPASS")
$1 = -7
(gdb) call passwd("The Pass")
Oh Yeah!
$2 = 9
```

Código 4.38. Invocação de função independente do contexto através do GDB.

4.6.2. Injeção de Código com `LD_PRELOAD`

Um recurso de modificação de fluxo de execução nativamente presente na plataforma Linux é o *Linux dynamic linker preload* (*LD_PRELOAD*). que permite que uma biblioteca especificada pelo usuário seja chamada com preferência sobre as bibliotecas originalmente referenciadas pelo binário, tais como a *libc*, de modo que funções presentes na biblioteca injetada que possuam a mesma assinatura (nome, parâmetros e retorno) serão preferencialmente invocadas pelo binário em execução, como exemplificado pelo Código 4.39. A técnica de injeção de código via `LD_PRELOAD` é utilizada tanto por atacantes como por analistas, como descrito a seguir. As assinaturas das funções referidas nesta seção podem ser encontradas nas *manpages* [Kernel.org 2017].

```
LD_PRELOAD=./library.so ./binary.bin
```

Código 4.39. Exemplo de chamada LD_PRELOAD na qual a biblioteca substitui funções originalmente referenciadas pelo binário

4.6.2.1. Contornando anti-análise com `LD_PRELOAD`

Na Seção 4.5.1, apresentamos alguns desafios de análise dinâmica de exemplares evasivos, dentre os quais o *delay* de execução via longas chamadas da função *sleep*. A injeção de código via *LD_PRELOAD* pode ser utilizada para contornar esse tipo de técnica de anti-análise através da substituição da função original por uma função modificada que ignore o tempo estipulado pelo atacante e retorne imediatamente, como exemplificado pelo Código 4.40.

```
unsigned int sleep(unsigned int seconds){
  return 0;
}
```

Código 4.40. Implementação alternativa da função sleep para contornar a evasão de análise via longos delays

Devido a precedência de execução da biblioteca injetada em relação a biblioteca original, a função *sleep* modificada será invocada pelo *malware* e a execução deste procederá normalmente, possibilitando a análise deste.

4.6.2.2. *Logging* usando `LD_PRELOAD`

A modificação do fluxo de execução através de *LD_PRELOAD* pode ser utilizada para se implementar mecanismos de *log* customizados, de forma semelhante a `strace` e `ltrace`, através da substituição das funções originais por funções que registrem a chamada da função. Para isso, deve se estabelecer uma função `trampolim` que realize este registro e, em seguida, retorne a execução para a função originalmente referenciada.

A Figura 4.6 exibe o fluxo de execução original de invocação de uma função a partir de um processo não monitorado, realizando a busca do endereço da função a ser executada na tabela de símbolos.

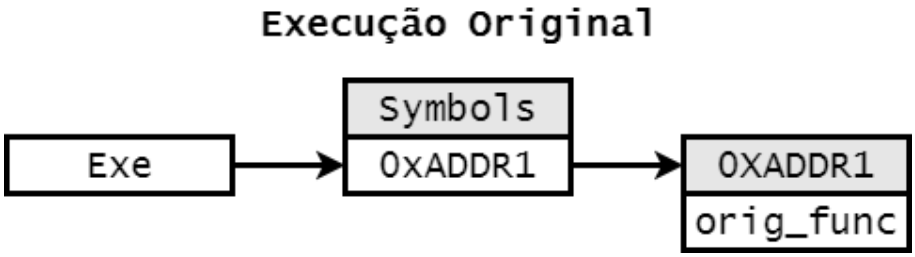


Figura 4.6. *Logging* com LD_PRELOAD. Fluxo original de execução através da invocação da chamada de função originalmente definida na tabela de símbolos.

A Figura 4.7, por sua vez, exibe o fluxo de execução alterado devido a injeção de código via *LD_PRELOAD*. Neste caso, uma função (trampolim) com a mesma assinatura da original é invocada previamente e realiza o registro das informações de *log*. Após o registro, a execução é direcionada para o endereço original para que a função monitorada realize a operação requisitada pelo binário.

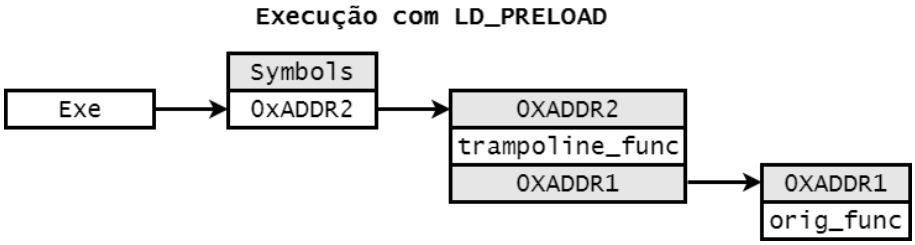


Figura 4.7. *Logging* com LD_PRELOAD. Fluxo de execução alterado após a modificação da tabela de símbolos por *LD_PRELOAD* para a invocação de uma função trampolim.

Ao contrário do caso da função *sleep*, em que ignorar o comportamento original da função é desejável para se mitigar evasões por *delay* da execução, no caso da implementação de um mecanismo de *logging*, deseja-se que a função original seja executada. Caso contrário, a execução do binário seria prejudicada. Desta forma, deve-se proceder a invocação da função original após o registro da operação, tal como exemplificado pelo Código 4.41. Note que esta estratégia requer que (i) a função trampolim identifique e armazene o endereço da função original e que (ii) a função original seja invocada com os mesmos parâmetros passados para a função trampolim.


```
typedef int (*orig_puts_f_type)(const char *str);

int puts(const char *str){
    FILE *fd = fopen("log","a+");
    orig_fprintf_f_type orig_fprintf;
    orig_fprintf = (orig_fprintf_f_type)dlsym(RTLD_NEXT,"fprintf");
    orig_fprintf(fd,str);
    fclose(fd);

    orig_puts_f_type orig_puts;
    orig_puts = (orig_puts_f_type)dlsym(RTLD_NEXT,"puts");
    return orig_puts(str);
}
```

Código 4.41. Código da função trampolim injetada via LD_PRELOAD para registrar a invocação da função. Após o registro, deve-se invocar a função original para garantir a realização da operação

4.6.2.3. *Rootkits* baseados em `LD_PRELOAD`

A capacidade de injeção de código provida pelo mecanismo *LD_PRELOAD* é explorada não apenas por analistas mas também por atacantes, sendo observada frequentemente na prática através de *rootkits*, um tipo de *malware* utilizado para esconder ou camuflar processos maliciosos do usuário [Beegle 2007], por exemplo, no caso de mineradores de cripto moedas [Remillano 2018].

O Código 4.42 exibe uma prova de conceito de um *rootkit* injetado via `LD_PRELOAD` na ferramenta *list directory contents* (*ls*), de modo a esconder um arquivo malicioso (*HIDDEN*) durante a leitura do diretório, de modo que um usuário seja incapaz de localizá-lo.

```
typedef struct dirent* (*orig_readdir_type)(DIR *dirp);
  struct dirent *readdir(DIR *dirp){
  struct dirent* valueOfReturn;
  orig_readdir_type orig_readdir;
  orig_readdir = (orig_readdir_type)dlsym(RTLD_NEXT,"readdir");
  valueOfReturn = orig_readdir(dirp);
  if(strcmp(HIDDEN,valueOfReturn->d_name) == 0)
    return NULL;
  return valueOfReturn;
}
```

Código 4.42. Exemplo de rootkit capaz de impedir a listagem de um arquivo.

4.6.2.4. Detectando injeções com `LD_PRELOAD`

Embora bastante poderoso, a injeção de código por `LD_PRELOAD` também pode ser contornada por atacantes, seja pela utilização de ligação estática, gerando, assim, um binário com tabela de símbolos vazia, impossibilitando a instrumentação, seja pela detecção direta da injeção de código no processo em execução, como exemplificado pelo Código 4.43. Este código detecta, através de execução (*getenv* ou */etc*), se as bibliotecas

de *LD_PRELOAD* estão presentes. Como contramedida, analistas podem se utilizar de instrumentação a nível de *kernel*, como discutido na Seção 4.8.

```
1  int main()
2  {
3      if(getenv("LD_PRELOAD"))
4          printf("LD_PRELOAD␣detected␣through␣getenv()\n");
5      else
6          printf("Environment␣is␣clean\n");
7      if(open("/etc/ld.so.preload", O_RDONLY) > 0)
8          printf("/etc/ld.so.preload␣detected␣through␣open()\n");
9      else
10         printf("/etc/ld.so.preload␣is␣not␣present\n");
11     return 0;
12 }
```

Código 4.43. Detecção da injeção de código via LD_PRELOAD

4.7. Demais Mecanismos de Monitoração em Modo Usuário

Além das soluções de análise especificamente voltadas a inspeção, monitoração e análise de binários, tais quais as anteriormente apresentadas, a plataforma `Linux` apresenta outras soluções que podem ser utilizadas para fins de análise em cenários específicos. Apresentamos, a seguir, o uso de monitores do sistema de arquivos e de mecanismos de *log* para fins de identificação de comportamentos maliciosos.

4.7.1. Monitoração do Sistema de Arquivos

Embora não sejam especificamente focados na análise e engenharia reversa de exemplares suspeitos, alguns mecanismos de monitoração podem ser empregados para este fim quando da operação em contextos específicos. Por exemplo, a monitoração do sistema de arquivos pode indicar atividades suspeitas relacionadas a criação e remoção de arquivos. Este tipo de monitoração pode ser utilizada, por exemplo, para a identificação de exemplares maliciosos pertencentes a famílias de *ransomware*, como o *Erebus* [Z. Chang 2017], que realizam acessos sequenciais aos arquivos do sistema para encriptá-los e demandar resgate.

A monitoração de ações no sistema de arquivo pode ser monitorada pelo *framework* `inotify` [McCutchan 2005], que permite a um administrador do sistema colocar vigias (*watches*) em arquivos e/ou diretórios de interesse. O Código 4.44 apresenta *logs* de monitoração via `inotify` de um diretório afetado pela execução do `Erebus`. Nota-se que o exemplar criar cópias criptografadas dos arquivos originais, sendo estes deletados. O uso da *API inotify* alerta o sistema ou usuário que a configurou em tempo real, de modo que se possa tomar uma possível ação de bloqueio ou de registro do evento em um *log* para posterior análise forense.


```
1 CREATE event: /home/sbseg/myfile.1
2 DELETE event: /home/sbseg/myfile
3 CREATE event: /home/sbseg/myfile2.1
4 DELETE event: /home/sbseg/myfile2
5 CREATE event: /home/sbseg/myfile3.1
6 DELETE event: /home/sbseg/myfile3
```

Código 4.44. Monitoração do diretório sbseg/ via inotify durante a execução do ransomware Erebus.

4.7.2. Mecanismos de *Logging*

Além da monitoração direta do sistema de arquivos via `inotify`, outros mecanismos de *log* presentes na plataforma `Linux` podem ser utilizado para monitorar os múltiplos subsistemas, o que pode incluir chamadas de sistema, comandos de usuário, eventos de segurança e acesso de rede. Uma das soluções que possibilita esta monitoração é o *Linux Audit system* (`Audit`) [Jahoda et al. 2017], sendo totalmente configurável a partir de regras. O Código 4.45 exemplifica o estabelecimento de regras para registrar acessos aos arquivos `passwd` e `shadow`.

```
1 auditctl -w /etc/passwd -p r
2 auditctl -w /etc/shadow -p r
```

Código 4.45. Definição de regra do Audit para monitoração da leitura dos arquivos passwd e shadow.

O `Audit` não provê mecanismos ativos para o bloqueio de ataques, se limitando a registrar os eventos ocorridos num sistema em um arquivo de *log* para consulta posterior. Apesar disto, este tipo de monitoração pode indicar claramente a ocorrência de diversos ataques. O Código 4.46, por exemplo, ilustra o *log* gerado pelo *audit* quando da monitoração da execução de um binário explorando a vulnerabilidade *DirtyCow* [Oster 2019]. Nota-se que, após realizar a exploração, o binário acessa os arquivos *passwd* e *shadow*, para a atribuição de permissão de super-usuário, e eleva seus próprios privilégios para *root* através do comando *su*, registrado através da invocação desta chamada de sistema.

```
1 type=PATH msg=audit(1565115273.377:158): item=0 name="/etc/passwd"
    inode=2105354 dev=08:01 mode=0100644 ouid=0 ogid=0 rdev=00:00
    nametype=NORMAL cap_fp=0000000000000000 cap_fi=0000000000000000
    cap_fe=0 cap_fver=0
2 type=PATH msg=audit(1565115273.377:159): item=0 name="/etc/shadow"
    inode=2105378 dev=08:01 mode=0100644 ouid=0 ogid=0 rdev=00:00
    nametype=NORMAL cap_fp=0000000000000000 cap_fi=0000000000000000
    cap_fe=0 cap_fver=0
3 type=SYSCALL msg=audit(1565115273.377:159): arch=c000003e syscall=2
    success=yes exit=5 a0=7f0a44c46c50 a1=80000 a2=1b6 a3=80000 items
    =1 ppid=2604 pid=4308 auid=4294967295 uid=1000 gid=1000 euid=0
    suid=0 fsuid=0 egid=1000 sgid=1000 fsgid=1000 tty=pts2 ses
    =4294967295 comm="su" exe="/home/sbseg/dcow" key=(null)
```

Código 4.46. Log do Audit coletado durante um ataque DirtyCow.

4.8. Mecanismos para Monitoração em *Kernel*

Quando atacantes desenvolvem exemplares de *malware* avançados a ponto de evadir os principais mecanismos de monitoração a nível de usuário, analistas devem recorrer a mecanismos de monitoração que operem em nível mais privilegiado que o objeto analisado, como o *kernel*, de modo a evitarem serem descobertos e subvertidos [Rossow et al. 2012]. Nesta seção, apresentamos os fundamentos da programação de módulos de *kernel* e da implementação de mecanismos para interposição de rotinas através destes. O objetivo desta seção não é ser um guia exaustivo para a modificação do *kernel*, mas sim uma introdução ao tema com foco em monitoração de aplicações. Informações detalhadas sobre a programação em *kernel* podem ser obtidas consultando a literatura especializada [Corbet et al. 2005].

4.8.1. Módulos de *Kernel*

Um *Loadable Kernel Module (LKM)* é um mecanismo para adicionar ou remover funcionalidades do *kernel* `Linux` em tempo de execução, sendo comumente usados por *drivers* de dispositivos, permitindo que o *kernel* se comunique com dispositivos de *hardware* sem possuir conhecimento prévios sobre o funcionamento destes. Ademais, LKMs também são frequentemente utilizados para implementar sistemas de arquivos e protocolos de rede, além de novas *features* no *kernel*, como *frameworks* de segurança [AppArmor 2019, Corporation 2015, Spengler 2019, Morris 2013b]. O uso de LKMs é vantajoso pois elimina a necessidade de se recompilar o *kernel* a cada alteração, reduz o tamanho total do código do *kernel* e também a complexidade das estruturas de controle responsáveis por gerenciar as novas funcionalidades. O código 4.8.1 apresenta o código fonte de um módulo simples, que utiliza das estruturas básicas do *kernel* para executar código em *kernel*. No caso, são exibidas mensagens de depuração no carregamento e descarregamento do módulo.

```
1  #include <linux/init.h>
2  #include <linux/module.h>
3  #include <linux/kernel.h>
4  MODULE_LICENSE("GPL");
5
6  static int __init lkm_init()
7  {
8    printk(KERN_INFO "Kernel!\n");
9    return 0;
10 }
11 static void __exit lkm_exit()
12 {
13   printk(KERN_INFO "Tchau!\n");
14 }
15
16 module_init(lkm_init);
17 module_exit(lkm_exit);
```

4.8.2. Suporte a Chamadas de Sistema

Em última instância, as chamadas de API originadas em espaço de usuário são entregues ao *kernel* na forma de chamadas de sistema (*system calls*). O *kernel* `Linux` mantém uma

tabela de apontadores que fazem referência às chamadas de sistema e exporta estes endereços para o espaço de usuário. Essa tabela de apontadores (*syscall table*) em conjunto com as *syscalls* formam a *syscall layer* do *kernel* (ou *syscall subsystem*). Detalhes da interação dos dados entre nível de usuário e *kernel* podem ser encontrados na documentação interativa do *kernel* [The Linux Kernel doc. 2017].

Tal qual apresentado para o mecanismo `LD_PRELOAD`, a tabela de chamadas de sistemas também pode ser modificada para a implementação de funções trampolim. O uso deste tipo de técnica em nível de *kernel* é denominada *syscall hooking* e é implementado através dos módulos LKM anteriormente apresentados. Dado que a tabela de chamadas de sistema é global, a implementação deste tipo de técnica em *kernel* se mostra vantajosa pois dispensa a necessidade de se realizar o *trace* de processos individuais em modo usuário, sendo, portanto, uma opção para a análise de exemplares evasivos. De forma análoga aos casos anteriores, este tipo de técnica também pode ser utilizada por atacantes para a implementação de *rootkits* de modo *kernel*.

4.8.2.1. Substituição da Tabela de Chamadas de Sistema

De um modo geral, as etapas requiridas para efetuar *syscall hooking* são as seguintes:

1. Localizar, na memória do *kernel*, a tabela de *syscalls*.
2. Localizar o deslocamento (*offset*) do apontador da tabela de *syscalls* para a função que executa a *syscall* originalmente presente no *kernel* e que será substituída.
3. Atribuir permissão de escrita ao segmento de memória que contém a tabela.
4. Substituir o apontador da *syscall* de interesse originalmente presente na tabela de *syscall* pelo apontador para a nova *syscall* contendo a função de interposição.
5. Remover a permissão de escrita da tabela.

Localizando a tabela de chamadas de sistema Em versões recentes do *kernel* `Linux`, como mecanismo de proteção, a tabela de chamadas de sistema não é mais exportada nativamente. Desta forma, seu endereço deve ser obtido de maneira alternativa. Para se localizar a tabela de *syscall*, usualmente promove-se uma varredura de uma região da memória conhecida até se localizar o símbolo para o código de uma *syscall*, a partir da qual pode-se resolver dinamicamente o símbolo da tabela de *syscalls* que lhe aponta. Alternativamente, pode-se também realizar o *parsing* do arquivo `/boot/System.map-{versão-kernel}`, que armazena todos os símbolos do *kernel* para fins de *debug*. Para fins didáticos, nos limitaremos a exemplificar apenas a primeira abordagem.

Em procedimentos de varredura, o programador inicialmente considera o endereço de um símbolo do *kernel* localizado próximo ou dentro da página de memória da *syscall table*, como *sys_open*. A partir deste, realiza a leitura de sequências de 4 ou 8 *bytes* (para ponteiros de 32 ou 64 *bits*) e compara os valores destes com os valores de apontadores para *syscalls* conhecidas até que estes coincidam. Tendo identificado esta região como sendo referente a página que armazena a tabela de chamadas de sistema, pode-se obter

o endereço desta considerando os cinco ponteiros anteriores a *sys_close* (contando com *sys_newstat*).

Uma implementação típica deste tipo de abordagem utiliza como ponteiro inicial o símbolo *loops_per_jiffy*, que se encontra sempre anterior ao apontador para a tabela de *syscall* na lista de símbolos do *kernel* e promove iteração de uma palavra do processador por vez, usando o número de indexação de uma das *syscalls*, e.g., o macro *__NR_close* para o símbolo de *sys_close*, como uma lista de ponteiros. Caso este ponteiro esteja apontando para o mesmo endereço de memória (*entry point*) de uma *syscall*, a *syscall table* foi encontrada. O Código 4.47 exemplifica esta abordagem.

```
1  unsigned long ptr;
2  unsigned long *p;
3  static long (*sys_close) (unsigned int fd)=NULL;
4
5  sys_close=(void *)kallsyms_lookup_name("sys_close");
6  if (!sys_close)
7  {
8    printk(KERN_DEBUG "[HOOK] Symbol sys_close not found\n");
9    return NULL;
10 }
11
12 for (ptr = (unsigned long)sys_close;
13   ptr < (unsigned long)&loops_per_jiffy;
14   ptr += sizeof(void *))
15   {
16     p = (unsigned long *)ptr;
17     if (p[__NR_close] == (unsigned long)sys_close)
18       {
19         printk(KERN_DEBUG "[HOOK] Found the sys_call_table!!!\n");
20         return (unsigned long **)p;
21       }
22   }
23 return NULL;
24 }
```

Código 4.47. Função find_sys_call_table() utilizada para identificar o endereço da tabela de chamadas de sistema.

Escrevendo na tabela de chamadas de sistema Após a localização do endereço da tabela de chamadas de sistema, deve-se proceder a substituição do apontador contida nesta pelo apontador para a função de interceptação desejada. Contudo, para fins de segurança, desde a versão *2.6.12* do *kernel*, a região de memória contendo a tabela é marcada como *read-only* através do *bit* `Protected Mode Enable` no registrador *CR0* das arquiteturas Intel, o que requer que esta proteção seja desligada através da atribuição de permissão de escrita para a página de memória contendo a tabela de modo a possibilitar a escrita do novo apontador.

Nas arquiteturas *i386* e *x86-64* da Intel, a proteção e gerência das áreas de memória do *kernel* são controladas pelos registradores *Control register (CR0-CR2)* do processador. Ao final do *boot* do *kernel*, assim que o subsistema de *syscalls* termina o carregamento e antes do subsistema de módulos iniciar, o *kernel* trocar o registrador *CR0* para *write-protect*, evitando que a *CPU* consiga escrever em áreas de memória marcadas como

leitura. A alteração da permissão de escrita é realizada através da escrita do *bit* "0" na posição "16" do registrador *CR0*. Para facilitar a manipulação de registradores, o *kernel* exporta alguns símbolos: *CR0_WP*, uma palavra do processador com o 16º *bit* como "1"; *write_cr0*, função que encapsula as operações dependentes de arquitetura para escrita diretamente no registrador `CR0`; *read_cr0*, função que encapsula operações para a leitura da palavra no registrador `CR0`.

O Código 4.48 ilustra um excerto do código de inicialização do módulo responsável por implementar o *hook* da *syscall sys_uname*, incluindo a troca da permissão da tabela de *syscall*, a escrita da *syscall* de interposição, a resolução dinâmica do *set_memory_rw* e a localização da tabela de *syscall*.

```
1  int ret; unsigned long cr0;
2
3  syscall_table = (void **)find_sys_call_table();
4  if (!syscall_table){
5    printk(KERN_DEBUG "[HOOK] Trying sys_call_table symbol\n");
6    syscall_table=(void **)kallsyms_lookup_name("sys_call_table");
7    if (!syscall_table)
8    {
9      printk(KERN_DEBUG "[HOOK] Cannot find the sys_call_table address\
         n");
10     return -EINVAL;
11   }
12 }
13
14 cr0 = read_cr0();
15 write_cr0(cr0 & (~CR0_WP));
16 do_set_memory_rw = (void *)kallsyms_lookup_name("set_memory_rw");
17 do_set_memory_ro = (void *)kallsyms_lookup_name("set_memory_ro");
18
19 if (do_set_memory_rw == NULL)
20 {
21   printk(KERN_DEBUG "[HOOK] Symbol not found: 'set_memory_rw'\n");
22   return -EINVAL;
23 }
24
25 ret = do_set_memory_rw(PAGE_ALIGN((unsigned long)syscall_table),1);
26 if(ret){
27   printk(KERN_DEBUG
28   "[HOOK] Cannot set the memory to rw (%d) at addr 0x%16lx\n",
29   ret, PAGE_ALIGN((unsigned long)syscall_table));
30   return -EINVAL;
31 }else
32   printk(KERN_DEBUG "[HOOK] Syscall Table page set to rw\n");
33
34 orig_sys_uname = syscall_table[__NR_uname];
35 syscall_table[__NR_uname] = hook_sys_uname;
36 write_cr0(cr0);
37 return 0;
38 }
```

Código 4.48. Função __init do módulo de kernel que implementa o hook da syscall sys_uname.

O vetores *orig_sys* e *hook_sys_uname*, armazenam, respectivamente, os apontadores para a *syscall* originalmente presente no *kernel* e para a função de interposição. O escaneamento de endereços (linha 8) é executado nos casos em que a versão do *kernel* é superior a *4.17* e, portanto, os símbolos das *syscalls* não são exportados. Nas linhas 38 e 39, o índice `NR_uname` é um inteiro referente a posição da *syscall* a ser substituída na tabela de chamadas de sistema—neste caso *sys_uname*. Desta forma, toda vez que a *syscall* `uname` for invocada, a função do módulo *hook_sys_uname* será chamada.

A função de interposição da *syscall* deve ter o mesmo protótipo usado pela *syscall* original, além de seguir o mesmo acesso aos *buffers* do usuário feita pela *syscall*. A implementação das *syscalls* originalmente presentes no *kernel* se encontram em *"kernel_source/kernel/sys.c"*, sobre a macro *SYSCALL_DEFINEN*, onde ***N*** é o número de parâmetros da *syscall*. O Código 4.49 apresenta a função de interposição da *syscall* `uname`, onde o acesso aos *buffers* é feito respeitando o mapeamento do *kernel* entre o nível de usuário. A linha 6 passa a execução para a *syscall* original, cujo ponteiro foi salvo na linha 38 do Código 4.48, onde apenas o *buffer* `sysname` é alterado para, de forma didática, ilustrar a substituição da chamada original.

```
1  asmlinkage long hook_sys_uname(struct new_utsname __user *name){
2    char msg[5]="Hook\0";
3    struct new_utsname tmp;
4    orig_sys_uname(name);
5    if(!copy_from_user(&tmp,name,sizeof(tmp)))
6    {
7      memcpy(tmp.sysname,msg,5);
8      if(copy_to_user(name,&tmp,sizeof(tmp)))
9        printk(KERN_DEBUG "[HOOK] Can't write to user-buffer!\n");
10   }
11   else
12     printk(KERN_DEBUG "[HOOK] Can't copy user-buffer :(\n");
13   return 0;
14 }
```

Código 4.49. Função de interposição.

O Código 4.50 ilustra a substituição da *syscall* através da saída do utilitário `Driver Message` (dmesg).

```
1  uname -s; rmmod hook; dmesg; uname -s
2  [ 8001.652808] [HOOK] Symbol sys_close not found
3  [ 8001.652808] [HOOK] Trying sys_call_table symbol
4  [ 8001.656808] [HOOK] System call table at 0xffffffff8e6001e0
5  [ 8001.660797] [HOOK] Syscall Table page set to rw
6  [ 8001.660797] [HOOK] sys_uname: 0xffffffff8da93460 - hook_sys_uname:
       0xffffffffc0860000
7  [ 8003.016080] [HOOK] Inside hook_sys_uname
8  [ 8003.016080] [HOOK] Can't write to user-buffer!
9  [ 8073.156170] [HOOK] Inside hook_sys_uname
10 [ 8073.156170] [HOOK] uname->sysname: Linux
11 [ 8073.291740] [HOOK] released module
12 Linux
```

Código 4.50. Exemplo de saída do dmesg durante o carregamento do LKM implementando o hook de syscall.

É importante notar que o uso frequente de *hooking* de funções do *kernel* `Linux` por diversas soluções motivou a criação do *Linux Security Modules* [Morris 2013a] e do SystemTap [Makarov 2019] para unificar e padronizar os *hooks* em *callbacks*. Entretanto, sistemas independentes ao *kernel mainstream*, tais como o `GrSecurity` [Spengler 2019] e códigos maliciosos, ainda utilizam *hooks* explícitos pela simplicidade e compatibilidade com outras *features* do *kernel*, possibilitando um maior nas operações do *kernel*, principalmente com *hooks* no subsistema de *syscall*.

4.9. Monitoração de Tráfego de Rede

Além das interações com o sistema operacional, parte relevante dos comportamentos exibidos por exemplares maliciosos se refere a interação com outros computadores via rede, de modo que a análise do tráfego de rede entre a máquina de análise e a rede remota (e.g., Internet) pode revelar informações importantes para o entendimento do exemplar desconhecido.

Técnicas para análise do tráfego de rede são bem descritas na literatura [Combs 2012] e nosso objetivo não é apresentar um guia exaustivo delas, mas apresentar as especificidades das análises na plataforma `Linux`. Mais especificamente, apresentamos, nesta seção, abordagens para se analisar exemplares suspeitos utilizando-se do filtro de pacotes `iptables` já presente nos sistemas baseados em `Linux`. Novamente, nosso objetivo não é ser um guia exaustivo de uso desta solução, o que pode ser consultado na literatura [Purdy 2004], mas sim apresentar uma introdução ao assunto.

O filtro de pacotes `iptables` atua em três cadeias: (i) `INPUT`, para as conexões originadas externamentes e que se destinam a máquina em questão; (ii) `FORWARD`, para conexões originadas externamente e que se destinam a outras máquinas, sendo roteadas através da máquina em questão; e (iii) `OUTPUT`, para conexões originadas localmente e destinadas a máquinas externas. Para cada uma destas cadeias, pode-se definir políticas de segurança, como a permissão (`ACCEPT`) ou o bloqueio (`DROP`) das conexões, como ilustrado pelo Código 4.51.

```
1 iptables -A INPUT -j ACCEPT
2 iptables -A FORWARD -j ACCEPT
3 iptables -A OUTPUT -j DROP
```

Código 4.51. Regras básicas iptables. O tráfego de saída pode ser bloqueado de modo a evitar que infecções se propaguem a partir da sandbox.

O bloqueio das conexões de saída pode ser utilizado para a implementação de um ambiente isolado da rede, como a *sandbox* referida na Seção 4.5, de modo a permitir a execução de exemplares maliciosos na máquina local sem que estes exemplares infectem as máquinas vizinhas e propaguem suas infecções.

Além de políticas globais para toda a cadeira, as regras `iptables` podem ser definidas para portas e protocolos específicos, como exemplificado pelo Código 4.52, de modo a permitir maior flexibilidade de operação.

```
1 iptables -A INPUT -p tcp -j ACCEPT
2 iptables -A INPUT -p udp -j DROP
```

Código 4.52. Exemplos de políticas iptables. Regras podem ser definidas para cada protocolo.

Para fins de análise, a principal política de interesse é o estabelecimento de registros para as conexões, de modo que se possa investigar com quais endereços IP e em quais portas o código sob análise se comunica. O Código 4.53 ilustra o estabelecimento de políticas de *log* padrão para todas as cadeias.

```
1 iptables -A INPUT -j LOG
2 iptables -A FORWARD -j LOG
3 iptables -A OUTPUT -j LOG
```

Código 4.53. Estabelecimento de políticas de log utilizando-se iptables para fins de monitoração.

Através da política de *log*, pode-se identificar alguns comportamentos típicos de exemplares maliciosos. O Código 4.54 apresenta um excerto de *log* de um exemplar malicioso do tipó *scanner*, que escaneia toda a rede ao qual o computador está conectado, de maneira sequencial, visando a identificação de vulnerabilidades que permitam a propagação do conteúdo malicioso do *malware*.

```
1 May 13 13:21:49 lab kernel: [ 3610.320968] IN= OUT=ens3 SRC=192
    .168.122.5 DST=91.189.89.196
2 May 13 13:21:49 lab kernel: [ 3610.321356] IN= OUT=ens3 SRC=192
    .168.122.5 DST=91.189.89.197
3 May 13 13:21:49 lab kernel: [ 3610.321503] IN= OUT=ens3 SRC=192
    .168.122.5 DST=91.189.89.198
4 May 13 13:21:49 lab kernel: [ 3610.321633] IN= OUT=ens3 SRC=192
    .168.122.5 DST=91.189.89.199
```

Código 4.54. Log de rede exemplificando a atuação de um malware do tipo scanner.

4.10. Conclusão

Nesse minicurso, apresentamos os conceitos básicos da engenharia reversa de aplicações maliciosas e introduzimos o leitor as características particulares das soluções de análise para o ambiente Linux e também aos comportamentos maliciosos dos exemplares afetando esta plataforma. Acreditamos que a partir dos conhecimentos obtidos neste curso, os leitores estarão aptos a realizar procedimentos básicos de análise em artefatos desconhecidos e a projetar experimentos de análise de binário. Esperamos, assim, contribuir para o desenvolvimento das pesquisas dos leitores em análise de binários de maneira geral. Embora focado na plataforma `Linux`, acreditamos que os conhecimentos de análise aqui apresentados possam estimular o desenvolvimento dos leitores quanto a análise de

binários em múltiplas plataformas (*Unix-Like*). Por fim, recomendamos a leitura dos trabalhos referenciados para que o leitor obtenha uma compreensão mais detalhada dos conceitos apresentados neste capítulo. Em especial, recomendamos a leitura da bibliografia especializada na plataforma `Linux` [Galante et al. 2018, Cozzi et al. 2018, Damri and Vidyarthi 2016, Isohara et al. 2010, Duncan and Schreuders 2019] para uma melhor compreensão de como as soluções apresentadas podem ser empregadas nas múltiplas tarefas associadas a engenharia reversa e segurança computacional.

Agradecimentos. Os autores agradecem a: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), em especial via Projeto FORTE - Forense Digital Tempestiva e Eficiente (Processo: 23038.007604/2014-69 - Edital 24/2014 - Programa Ciências Forenses); Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), em especial via Programa de Bolsas de Doutorado (processo número 164745/2017-3) e Programa de Bolsas de Iniciação Científica (processo número 138239/2017-7).

Referências

[Afonso et al. 2015] Afonso, V. M., de Amorim, M. F., Grégio, A. R. A., Junquera, G. B., and de Geus, P. L. (2015). Identifying android malware using dynamically obtained features. *Journal of Computer Virology and Hacking Techniques*, 11(1):9–17.

[Alves et al. 2017] Alves, P., Brobecker, J., Evans, D., and Zaretskii, E. (2017). Gdb: The gnu project debugger. `https://www.gnu.org/software/gdb/`.

[Android 2017] Android (2017). System and kernel security. `https://source.android.com/security/overview/kernel-security.html#linux-security`. Acessado em: Abril/2017.

[AppArmor 2019] AppArmor, W. (2019). Apparmor project wiki. `http://wiki.apparmor.net`.

[Beegle 2007] Beegle, L. E. (2007). Rootkits and their effects on information security. *Inf. Sys. Sec.*, 16(3):164–176.

[Botacin et al. 2018] Botacin, M. F., de Geus, P. L., and Grégio, A. R. A. (2018). The other guys: automated analysis of marginalized malware. *Journal of Computer Virology and Hacking Techniques*, 14(1):87–98.

[Branco et al. 2012] Branco, R. R., Barbosa, G. N., and Neto, P. D. (2012). Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. `https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_WP.pdf`.

[C-Jump 2017] C-Jump (2017). 7. eflags individual bit flags. `http://www.c-jump.com/CIS77/ASM/Instructions/I77_0070_eflags_bits.htm`.

[Cheng et al. 2018] Cheng, B., Ming, J., Fu, J., Peng, G., Chen, T., Zhang, X., and Marion, J.-Y. (2018). Towards paving the way for large-scale windows malware analysis:

Generic binary unpacking with orders-of-magnitude performance boost. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 395–411, New York, NY, USA. ACM.

[Combs 2012] Combs, G. (2012). *Wireshark Network Analysis (Second Edition): The Official Wireshark Certified Network Analyst Study Guide*. Laura Chappell University.

[Coogan et al. 2009] Coogan, K., Debray, S., Kaochar, T., and Townsend, G. (2009). Automatic static unpacking of malware binaries. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, WCRE '09, pages 167–176, Washington, DC, USA. IEEE Computer Society.

[Corbet et al. 2005] Corbet, J., Rubini, A., and Kroah-Hartman, G. (2005). *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc.

[Corporation 2015] Corporation, N. D. (2015). About tomoyo linux. `http://tomoyo.osdn.jp/about.html.en`.

[Cozzi et al. 2018] Cozzi, Graziano, Fratantonio, and Balzarotti (2018). Understanding linux malware. `http://www.s3.eurecom.fr/~yanick/publications/2018_oakland_linuxmalware.pdf`.

[Damri and Vidyarthi 2016] Damri, G. and Vidyarthi, D. (2016). Automatic dynamic malware analysis techniques for linux environment. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 825–830.

[Duncan and Schreuders 2019] Duncan, R. and Schreuders, Z. C. (2019). Security implications of running windows software on a linux system using wine: a malware analysis study. *Journal of Computer Virology and Hacking Techniques*, 15(1):39–60.

[eliben 2017] eliben (2017). Parsing elf and dwarf in python. `https://github.com/eliben/pyelftools`.

[Fruhlinger 2018] Fruhlinger, J. (2018). The mirai botnet explained: How teen scammers and cctv cameras almost brought down the internet. `https://bit.ly/2Irz5e3`.

[Galante et al. 2018] Galante, L., Botacin, M., Grégio, A., and de Geus, P. L. (2018). Malicious linux binaries: A landscape. In *Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 213–222, Porto Alegre, RS, Brasil. SBC.

[Gebai and Dagenais 2018] Gebai, M. and Dagenais, M. R. (2018). Survey and analysis of kernel and userspace tracers on linux: Design, implementation, and overhead. *ACM Comput. Surv.*, 51(2):26:1–26:33.

[GNU 2018] GNU (2018). magic - linux man pages. `https://www.systutorials.com/docs/linux/man/5-magic/`.

[Grégio et al. 2012] Grégio, A. R. A., Afonso, V. M., Filho, D. S. F., de Geus, P. L., Jino, M., and dos Santos, R. D. C. (2012). Pinpointing malicious activities through network and system-level malware execution behavior. In Murgante, B., Gervasi, O., Misra, S., Nedjah, N., Rocha, A. M. A. C., Taniar, D., and Apduhan, B. O., editors, *Computational Science and Its Applications – ICCSA 2012*, pages 274–285, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Grégio et al. 2015] Grégio, A. R. A., Afonso, V. M., Filho, D. S. F., Geus, P. L. d., and Jino, M. (2015). Toward a Taxonomy of Malware Behaviors. *The Computer Journal*, 58(10):2758–2777.

[Isohara et al. 2010] Isohara, T., Takemori, K., Miyake, Y., Qu, N., and Perrig, A. (2010). Lsm-based secure system monitoring using kernel protection schemes. In *2010 International Conference on Availability, Reliability and Security*, pages 591–596.

[Jahoda et al. 2017] Jahoda, M., Krátký, R., Prpič, M., Čapek, T., Wadeley, S., Ruseva, Y., and Svoboda, M. (2017). A guide to securing red hat enterprise linux: System auditing. `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing`.

[Kernel.org 2017] Kernel.org (2017). The linux man-pages projects. `https://www.kernel.org/doc/man-pages/`.

[Koret and Bachaalany 2015] Koret, J. and Bachaalany, E. (2015). *The Antivirus Hacker's Handbook*. Wiley Publishing, 1st edition.

[Makarov 2019] Makarov, S. (2019). Systemtap overview. `http://sourceware.org/systemtap`.

[Matloff and Salzman 2008] Matloff, N. and Salzman, P. J. (2008). *The Art of Debugging with GDB, DDD, and Eclipse*. No Starch Press, San Francisco, CA, USA.

[McCutchan 2005] McCutchan, J. (2005). inotify - monitoring filesystem events. `http://man7.org/linux/man-pages/man7/inotify.7.html`.

[M.F.X.J. Oberhumer 2018] M.F.X.J. Oberhumer, László Molnár, J. F. R. (2018). Upx the ultimate packer for executables. `https://upx.github.io/`.

[Morris 2013a] Morris, J. (2013a). Linux security module framework. `https://www.linux.com/learn/overview-linux-kernel-security-features`.

[Morris 2013b] Morris, J. (2013b). Selinux project wiki. `http://selinuxproject.org`.

[O'Neill 2016] O'Neill, R. E. (2016). *Learning Linux Binary Analysis*. Packt Publishing.

[Oster 2019] Oster, P. (2019). Cve-2016-5195. `https://nvd.nist.gov/vuln/detail/CVE-2016-5195`.

[Peterson and Brown 1961] Peterson, W. W. and Brown, D. T. (1961). Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235.

[Purdy 2004] Purdy, G. N. (2004). *Linux iptables Pocket Reference*. O'Reilly.

[Remillano 2018] Remillano, A. I. (2018). Cryptocurrency-mining malware targets linux systems, uses rootkit for stealth. `https://tinyurl.com/y3yyv5oo`.

[Rossow et al. 2012] Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., and Steen, M. v. (2012). Prudent practices for designing malware experiments: Status quo and outlook. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 65–79, Washington, DC, USA. IEEE Computer Society.

[S. Weyergraf 2015] S. Weyergraf, S. B. (2015). Ht editor. `http://hte.sourceforge.net/index.html`.

[Sikorski and Honig 2012] Sikorski, M. and Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition.

[Simmonds 2015] Simmonds, C. (2015). *Mastering Embedded Linux Programming*, chapter Linking with libraries: static and dynamic linking. Packt Publishing.

[Skoudis and Zeltser 2003] Skoudis, E. and Zeltser, L. (2003). *Malware: Fighting Malicious Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Spengler 2019] Spengler, B. (2019). Grsecurity features. `https://grsecurity.net/features.php`.

[Tasiopoulos and Katsikas 2014] Tasiopoulos, V. G. and Katsikas, S. K. (2014). Bypassing antivirus detection with encryption. In *Proceedings of the 18th Panhellenic Conference on Informatics*, PCI '14, pages 16:1–16:2, New York, NY, USA. ACM.

[The Linux Kernel doc. 2017] The Linux Kernel doc. (2017). Linux system calls implementation. `https://linux-kernel-labs.github.io/master/lectures/syscalls.html`.

[Vecchia and Coral 2014] Vecchia, E. D. and Coral, L. (2014). Linux remote evidence colector – uma ferramenta de coleta de dados utilizando a metodologia live forensics. *Anais do SBSEG 2014*, pages 586–597.

[Venezla 2012] Venezla, P. (2012). A world without Linux: Where would Apache, Microsoft – even Apple be today? `http://www.infoworld.com/article/2616083/data-center/a-world-without-linux--where-would-apache--microsoft----even-apple-be-today-.html`. Acessado em: Abril/2017.

[Wang et al. 2018] Wang, D., Ming, J., Chen, T., Zhang, X., and Wang, C. (2018). Cracking iot device user account via brute-force attack to sms authentication code. In *Proceedings of the First Workshop on Radical and Experiential Security*, RESEC '18, pages 57–60, New York, NY, USA. ACM.

[Wong 2018] Wong, R. (2018). *Mastering Reverse Engineering: Re-engineer your ethical hacking skills*. Packt.

[Yocom et al. 2004] Yocom, N., Turner, J., and Davis, K. (2004). *The Definitive Guide to Linux Network Programming (Expert's Voice)*. Apress.

[Z. Chang 2017] Z. Chang, G. Sison, J. J. (2017). Erebus resurfaces as linux ransomware. `https://tinyurl.com/y6qwxs3q`.

SBSeg 2019

XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais

2 a 5 de Setembro – São Paulo, SP

sbseg2019.ime.usp.br

Patrocínio Diamante

nic.br – Núcleo de Informação e Coordenação do Ponto BR

cgi.br – Comitê Gestor da Internet no Brasil

CAPES

FAPESP

Atos

Patrocínio Ouro

CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico

Patrocínio Prata

IME-USP

INCT InterSCity

Patrocínio Bronze

PagSeguro

CLAVIS segurança da informação

Realização

SBC – Sociedade Brasileira de Computação

USP

IME-USP

Apoio

taqui