Capítulo

8

Garantia de Segurança em aplicações: De Requisitos a Padrões Seguros

Rodrigo Elia Assad, Felipe Silva Ferraz e Silvio Romeiro Lemos Meira

Abstract

It is historically known that extensive phases of rework affect directly both costs and quality of already developed applications. In regard to requirements, systems security figure among those which are left behind, generating undesired steps of reformulation. This article is a short presentation on design patterns and non functional security requirements; it relates both and shows that an initial use of certain patterns, in early development phases, might reduce the impacts of relegating the systems security requirement implementation to a second plan.

Resumo

Historicamente, sabe-se que extensas fases de manutenção ou alteração afetam diretamente os custos e a qualidade das aplicações desenvolvidas. Muitos destes problemas estão associados à negligência na elaboração dos requisitos, neste cenário Segurança dos Sistemas figura entre aqueles que mais são deixados em segundo plano, gerando indesejáveis etapas de reformulação. Por outro lado, quando se trata do envolvimento com Padrões de Projeto *esses aparecem como umas das principais respostas para a solução de problemas relacionados ao desenvolvimento. Diante disto, esse trabalho propõe uma relação entre Padrões de Projeto e Requisitos de Segurança de modo a, uma vez adotado nas etapas iniciais do desenvolvimento, diminuir os impactos relacionados à implementação de segurança de software a segundo plano.*

1.1. Segurança da Informação

O termo segurança é normalmente utilizado para definir técnicas utilizadas para minimizar as vulnerabilidades de bens e recursos, onde bens são qualquer coisa de

valor. Vulnerabilidade é qualquer fraqueza que possa ser explorada para se violar o sistema ou as informações que este contem ou os serviços que este provê[I. 7498-2, 1989]. Segurança depende mais do que apenas da integridade do software e dos mecanismos de proteção, ela também depende da própria configuração e uso do software[Zerkle, 1996] em si, segurança está também relacionada com a proteção aos acessos, manipulação intencional ou não de valores confidenciais armazenados no sistema bem como sua utilização não autorizada por terceiros.

Existem três conceitos básicos de relacionados a temas de Segurança de maneira geral, são eles: Confidencialidade, Integridade e Disponibilidade. Já os conceitos que estão relacionados diretamente com as pessoas que utilizam as informações ou serviços que os sistemas provêem são: Autenticação, Autorização e Não-Repudiação.

Quando uma informação é acessada ou copiada por alguém sem autorização para tal, o resultado é conhecido por perda de confidencialidade. Exemplos de bens que podem ser relacionados à confidencialidade podem incluir desde dados de pesquisa, como informações de usuários ou até mesmo valores bancários. Por outro lado, quando temos a alteração de uma informação disponível em uma rede de forma insegura ou em algum sistema inseguro, podemos ter que lidar com perdas de integridade que significa que modificações não autorizadas foram feitas na informação. Integridade é particularmente importante para setores como o financeiro, controle de trafego aéreo e controle financeiro. Por fim quando informações são apagadas ou se tornam inacessíveis, o resultado é a perda de disponibilidade, isso significa que pessoas autorizadas a ter acesso à determinada informação não conseguem acessá-la.

Diante desse cenário, tem sido firmada a existência de três etapas necessárias para se garantir a segurança de sistemas ou aplicações são elas: autenticação, autorização e auditoria[Samarati 2000].

Essencialmente essas etapas visão garantir que os princípios definidos anteriormente sejam respeitados e que dessa forma possa-se falar em segurança da informação. Cada uma dessas faz usos de uma, ou uma combinação, de métodos ou práticas para tentar garantir que o mal usou e ou acessos indevidos, não sejam executados. Entre eles podemos citar desde o estabelecimento de normas para senhas, como conceitos de criptografia moderna e até mesmo soluções de padrões de segurança.

1.1.1 Autenticação

Desde sua criação, e posterior popularização, a Internet sempre se mostrou com um mecanismo de anonimato, onde pessoas poderiam passar-se por outras ou até mesmo não passar por ninguém. Peter Steiner[Steiner 1993], Figura 1, satirizou essa situação com um cartoon que ficaria conhecido como uma das motivações para criação e definição de mecanismos onde esse tipo de situação pudesse ser combatido.

Figura 1: "Na Internet, ninguém sabe que você é um cachorro"[Steiner 1993].

Autenticação é o processo onde uma pessoa ou programa de computador prova a sua identidade com o intuito de obter alguma informação ou executar alguma ação[Kunyu 2009].

A identidade de uma pessoa é uma simples abstração, um identificador em uma aplicação específica, por ex. Provar, ou validar, consiste na parte mais importante desse conceito, é geralmente feita através de uma informação conhecida, como uma senha, ou palavra chave, ou um cartão com identificação visual, ou algo único com relação a sua aparência, impressões digitais, íris ou até mesmo amostra de DNA.

Um sistema de autenticação pode ser considerado forte, se fizer uso de pelo menos 2 dos 3 tipos de autenticações disponíveis.

A fraqueza desse tipo de sistema gira em torno da transmissão de tais informações através de canais, senhas podem ser roubadas, acidentalmente reveladas, ou até mesmo esquecidas.

1.1.2 Autorização

Autorização é o processo de dar permissão para fazer ou ter algo. Em Sistemas de múltiplos usuários um administrador de sistemas definirá quais usuários terão acesso e quais privilégios de execução esses terão[Schumacher 2006]. Considerando que alguém conseguiu se Autenticar em um sistema, o sistema pode necessitar identificar quais recursos a entidade poderá receber durante a sessão atual. O mecanismo de autorização é muita vezes visto como composto de dois momentos[Wang2009]:

- Ato de atribuir permissões ao usuário do sistema no momento de seu cadastro, ou posterior.
- Ato de checar as permissões que foram atribuídas ao usuário no momento de seu acesso.

Logicamente, o mecanismo de autorização vem posteriormente ao de Autenticação.

Assumindo agora que Autorização é no mínimo tão importante quanto Autenticação, faz-se necessário estender o escopo das soluções de segurança, que por ventura estejam relacionadas à Autenticação, para tentar solucionar também problemas de Autorização.

1.1.3 Auditoria

O processo de auditar um sistema é um dos mais importantes mecanismos de segurança que se pode fazer uso na hora de examinar, verificar e corrigir o funcionamento geral de um sistema. Conceitualmente o processo de auditar um sistema verifica se o mesmo está executando suas funções corretamente. Esse tipo de verificação é extremamente importante para processos de segurança, afinal não podemos considerar suficiente, apenas, implementar mecanismo de segurança em determinados ambientes, precisamos disponibilizar um mecanismo onde se possa verificar que os mecanismos realmente funcionam[Schumacher 2006][Dhillon 2008].

Existem duas etapas básicas de Auditoria, que na verdade, são apenas variações do mesmo conceito. A primeira forma trata do registro de mudanças no estado do sistema, ou seja, eventos e/ou mudanças de estados são registrados. A segunda forma seria um processo sistemático que examina e verifica o sistema, trata-se de uma consulta ao ambiente para avaliação do funcionamento do mesmo, ou seja, verifica-se se o sistema está funcionando de forma correta. Para um processo de auditoria ser eficiente, ambas as formas mencionadas precisam estar construídas e sintonizadas, é impossível executar uma avaliação sistemática de um sistema ou de um evento se esses não foram registrados, e mais, se os estados do sistema não forem gravados, não há a menor necessidade de se guardar, também, as mudanças no sistema, assim vamos considerar que para uma auditoria ser satisfatória esta deverá:

- Registrar os estados do sistema;
- Verificar, sistematicamente os estados armazenados[Dhillon 2008].

- Permitir que os registros sejam checados[Schumacher 2006].

Em linhas gerais, uma vez concluídas a etapa de Autenticação, onde se identifica o usuário e a etapa de Autorização, onde se atribuem às permissões do usuário, a etapa de Auditoria consiste em uma etapa mais longa, presente durante o ciclo de vida do sistema, ela vem para monitorar o uso do sistema, servindo, entre outras, como input para futuras ações, que podem incluir desde melhorias nas etapas anteriores até reestruturações do ambiente.

1.1.4 Não-Repudiação

Uma vez que temos um processo de Autenticação onde o usuário prova quem ele é, posteriormente temos o processo de Autorização onde este recebe suas permissões junto ao sistema, por fim temos os mecanismos de auditoria para monitorar o uso desse sistema registrando ações tomadas pelos usuários para posterior checagem. Sendo assim, uma vez validados o usuário e suas permissões e suas ações tendo sido registrada, não é factível que existem mecanismo que permitam a uma entidade autenticada negar suas ações. Nesse cenário temos o termo Não-Repudiação.

Não-Repudiação pode ser definido como um mecanismo ou serviço que provê provas da integridade e origem de um dado ambos de forma inegável e não criável, que pode ser verificado por uma terceira parte. Um serviço que pode atestar uma relação entre dado e autor do dado de forma genuína[Peris 2008][Adikari 2006]

1.2. Requisitos de Segurança e Padrões de Projeto

1.2.1.Padrões de Projeto e a Gangue dos Quatro

O entusiasmo, dos desenvolvedores de sistema, criado ao redor dos padrões é algo inquestionável desde a publicação do livro da conhecida, Gangue dos quatro ou Gang-of-Four. Desenvolvedores do mundo todo se uniram ao redor da idéia dos padrões na certeza de que esses permitiram uma maior clareza, versatilidade, direção e facilidade na escrita de códigos dessa forma, padrões conseguiram trilhar seu caminho para fazer parte de inúmeros projetos[Schumacher, 2006].

Para os autores do GoF[Gamma, 1994] padrões de projeto variam de acordo com a granularidade deles e nível de abstração, por conta dessa grande variação fez-se necessário organizá-los e categorizá-los. A classificação proposta auxilia no aprendizado do padrão e na forma de achar o padrão mais adequado para cada problema.

A classificação proposta levou em consideração dois critérios. O primeiro chamando propósito reflete o que o padrão faz. Podem ser de criação, estrutural ou comportamental. Padrões de criação tratam do processo de criação dos objetos. Padrões estruturais lidam na forma como as relações entre os objetos são criadas e por fim padrões de comportamento caracterizam a forma como objetos interagem e distribuem responsabilidades.

O segundo critério, chamado de Escopo, trata de onde o padrão é aplicado se a classes ou os objetos. Padrões de Classe lidam com relacionamentos entre as classes e suas subclasses. Padrões de Objetos lidam com relações que podem ser mudadas em tempo de execução e são mais dinâmicos.

Para os fins desse trabalho detalharemos mais os a classificação relacionada ao propósito.

1.2.1.1 Padrões de Criação.

Padrões de Projeto de criação são responsáveis por abstrair o processo de criação de determinados objetos. Auxiliam a deixar o sistema de forma independente de como os objetos serão criados, compostos ou representados. Uma entidade que representa o padrão de criação usará os princípios de OO como herança para variar os tipos de classe que ela criará[Gamma 1994].

Existem dois temas principais quando se fala em padrões de criação, no primeiro temos que eles encapsulam os conhecimentos referentes a que classe, concretamente, o sistema utiliza, e no segundo temos que eles escondem como, as lógicas e regras de negocio utilizadas, as instancias desses objetos são criadas e organizadas de forma a serem utilizadas. De forma geral para o sistema, tudo que é conhecido é apenas a interface definida, o chamado contrato. Conseqüentemente os padrões de criação fornecem uma maior flexibilidade com relação ao *o que* é criado, *quem* o cria, *como* este é retornado e *quando.*

Dessa forma os objetos retornados e utilizados pelo o sistema serão compostos de valores e comportamentos que poderão ser, facilmente, alterados em qualquer momento do desenvolvimento.

Alguns padrões de Criação do GoF:

Abstract Factory: Disponibiliza uma interface para criação de uma família de objetos, relacionados ou dependentes, sem ser necessária uma especificação concreta, ou seja, não é necessário informar qual classe precisa ser criada.

Uma hierarquia que encapsula diversas possibilidades de plataformas, ou formatos, e/ou um conjunto possível de produtos.

Builder: Separa a construção de um objeto complexo da sua representação, da sua classe definida, de modo que a mesma construção possa ser usada por diferentes representações.

Possibilita a criação de entidades complexas através da criação de variados tipos que terminarão por representar tal entidade.

Factory Method: Define um mecanismo para criar um objeto, um construtor virtual, deixando que a implementação saiba e decida, qual classe será utilizada para instanciar o mesmo. Normalmente consiste em um método estático de uma classe responsável por retornar um novo objeto dessa classe.

Difere-se principalmente da Abstract Factory por retornar apenas um tipo de objeto, enquanto que a Abstract está preparada para retornar uma família de objetos.

Prototype: Define os tipos de objetos a serem criados usando uma instância padrão, cria os novos objetos copiando o protótipo. Trabalho junto com uma determinada instancia de uma classe para criar instâncias futuras.

Singleton: Garante que apenas uma instância de uma determinada classe esteja disponível no sistema provê uma interface de acesso único para esta. Encapsula uma estrutura de inicialização *Just-in-time,* apenas na hora, ou inicialização no primeiro uso.

1.2.1.2 Padrões de Estrutura.

O foco dos Padrões de Estrutura, ou Estruturais, é definir como as classes e objetos são combinados de forma a criar sistemas maiores e mais complexos. Um exemplo simples é como combinar duas ou mais classes que já sejam produtos de uma herança em uma terceira. O resultado dessa estrutura é uma classe que combina as propriedades das duas classes pais originais, conceitualmente esse padrão é particularmente útil para fazer com que classes independentes possam trabalhar juntas[Gamma 1994].

Indo além dos padrões de projeto, trabalhar com estruturas significa prover novas funcionalidades aproveitando-se os comportamentos de diferentes classes já criadas.

Alguns padrões de Estrutura do GoF:

Adapter: Converte a interface de uma determinada classe em outra. Um Adapter transforma a classe alvo em uma forma que o cliente que a utilizará possa entender. Permite que diferentes classes possam interagir através do uso das novas interfaces criadas pelo adapter.

Bridge: Separa uma abstração da sua implementação de modo que ambos possam trabalhar de forma independente.

Composite: Compõe objetos em estruturas de árvore para representar uma hierarquia do tipo todo-parte. Composite permite que os clientes, utilizadores das entidades, utilizem os objetos individuais ou combinações deles da mesma forma.

Decorator: Atribui responsabilidades dinamicamente a um objeto. Decorators fornecem uma alternativa flexível a uma sub-classe para que essa consiga ter mais funcionalidades.

Facade: Fornece uma interface única, e unificada, para um conjunto de interfaces em um subsistema. Facade Define uma interface de alto nível que permite que o sistema seja mais fácil de ser utilizado, por fornecer apenas um ponto único de acesso.

1.2.1.3 Padrões de Comportamento.

O foco dos padrões de comportamento gira em torno dos algoritmos e atribuições de responsabilidades e comportamentos entre objetos, descrevem não apenas um padrão de objetos ou classes, mas também um padrão para comunicação entre eles. Esses Padrões caracterizam complexos fluxos de controle que normalmente são complicados de ser acompanhado em tempo de compilação. Eles buscam tirar o foco do desenvolvedor do

fluxo de controle para que ele possa se concentrar apenas na forma como os objetos estão interligados.

Alguns Padrões de Comportamento do GoF:

Iterator: Provê um mecanismo para acessar os elementos dentro de um conjunto de objetos, sistema, sem expor suas representações.

Observer: Define uma dependência 1-N (um para N) entre objetos, de forma que quando determinada ação ocorra em um objeto este tenha uma maneira de notificar, os N objetos, do evento ocorrido.

Encapsula o núcleo dos componentes em uma abstração e varia os componentes de modo a possibilitar diferentes notificações.

Command: Encapsula as requisições a um objeto permitindo, dessa forma, que o desenvolvedor possa parametrizar os clientes com diferentes tipos de requisição.

Memento: Sem violar o encapsulamento, captura e retorna ao exterior, o estado interno de um objeto, de modo que o objeto em questão possa ser retornado ao seu atual posteriormente.

Strategy: Define uma família de algoritmos, encapsula cada um deles, e faz com que eles possam ser intercambiáveis. Strategy permite que os algoritmos possam variar independentemente do cliente que o usa.

1.2.1.4 Abordagens a Padrões de Projeto

Existem diversos estudos focados em padrões de projeto. Por um lado temos estudos como o de Yoder[Yoder 1998] que foca em padrões de segurança mas afirma que padrões de projeto auxiliam na definição de arquiteturas de softwares, por outro temos estudos como os apresentados por Khomh[Khomh 2008][Haley 2004] onde ele mostra que padrões de projeto nem sempre se apresentam como uma boa solução do ponto de vista da qualidade de softwares por impactar negativamente em uma serie de critérios de qualidade.

Diferente desses, trabalhos como o de Sandhu tentam mostrar que Design Patterns, ou Padrões de Projeto, podem acelerar o processo de desenvolvimento de software oferecendo passos, pré-testados varias vezes, a serem seguido. Design Patterns mostram sua eficiência em todas as fases desde o desenvolvimento até a manutenção e podem ser analisados pela sua qualidade, fator esse que é um dos aspectos mais relevantes na avaliação de qualquer estrutura de software. Sandhu[28] faz uma analise de um conjunto de métricas que podem ser aplicáveis e válidas para todos os padrões. Sua proposta foca principalmente em analisar a quantidade de padrões que foram adotados e utilizados em um determinado projeto, para que através da detecção de um conjunto desses padrões o seu framework proposto possa indicar um valor relacionado a uma de suas métricas que por sua vez são relacionadas à qualidade do sistema de maneira geral.

Já Outros trabalhos como o de Mario Luca e Giuseppe mostram que Padrões de projeto apresentam uma serie de problemas relacionados à qualidade do código gerado,

afetando de forma negativa métricas como modularidade. Nesse trabalho é apresentada uma abordagem onde podemos fazer ver re-implementações de padrões de projeto através de orientação a aspectos de uma forma bastante efetiva de modo a diminuir tais impactos negativos. É-nos mostrado que orientação a aspectos permite criar diferentes soluções para os problemas citados mesmo que estas gerem novos problemas como coesão e acoplamento, inerentes a próprio Orientação a Aspectos.

Rudzki[Lutz 2000] foca principalmente em como o uso de design patterns na montagem de interfaces remotas influencia na performance de aplicações distribuídas. Os padrões estudados por Rudzki são os considerados como boa opções para design de aplicações.

Para Rudzki performance é apenas um entre muitos atributos de qualidade que podem ser utilizados para descrever as propriedades de uma aplicação, mas em muitos casos tais propriedades são colocadas em uma posição alta de listas de prioridades relacionadas a requisitos funcionais. Dessa forma qualquer medida que possa ser tomada durante o processo de desenvolvimento que auxilie na melhora da performance é benéfico uma vez que seu trabalho concluiu que decisões relacionadas ao uso de um determinado design pattern em um sistema distribuído impacta na performance do sistema como um todo.

Analisados de forma positiva ou negativa, é fácil perceber que existem um série de estudos focados em padrões de projeto, sejam analisando seu impacto ou aprovação, tais estudos já fazem parte do dia a dia dos desenvolvedores e equipes de software. Indo alem, já do ponto de vista de requisitos não funcionais, podemos verificar que performance da mesma forma que segurança aparece entre os requisitos não funcionais relativos a qualidade e como aqueles onde se perceber a maior negligência por parte de membros das equipes.

Em virtude dessas considerações nosso próximo estudo será conduzido com relação a requisitos de segurança, focando principalmente em seu entendimento.

1.2.2. Requisitos de Segurança

A engenharia de requisitos dentro de um negócio, sistemas, aplicações e componentes é muito mais do que apenas documentar os requisitos funcionais destes. Mesmo que, grande parte dos analistas se dedique a elicitar alguns requisitos de qualidade como: interoperabilidade, disponibilidade, performance, portabilidade e usabilidade, muitos deles ainda pecam no que diz respeito a analisar questões relacionadas a Segurança.

Infelizmente, documentar requisitos de segurança específicos é difícil. Estes tendem a se mostrar como de alto impacto para muitos requisitos funcionais. E mais, requisitos de segurança são, normalmente, expressados de forma a documentar os termos de como alcançar segurança a não na forma do problema que precisa ser resolvido[Haley 2004].

A maior parte dos analistas de requisitos não tem conhecimentos na área de Segurança, os poucos que receberam algum tipo de treinamento tiveram apenas uma visão geral sobre alguns mecanismos de segurança como senhas e criptografia ao invés de conhecer reais requisitos nessa área[Yoder 1996][Firesmith 2003][Firesmith 2004]

Requisitos de segurança tratam de como os bens de um sistema devem ser protegidos contra qualquer tipo de mal[Haley 2004][Haley 2006]. Um bem é algo no contexto do sistema, tangível ou não, que deve ser protegido[ISSO IEC]. Um mal ou ameaça da qual um sistema precisa ser protegido, é uma possível vulnerabilidade que pode atingir um bem. Uma vulnerabilidade é uma fraqueza de um sistema que um ataque tende a explorar. Requisitos de segurança são restrições nos requisitos funcionais com o intuito de reduzir o escopo das vulnerabilidades[Haley 2004].

Donald Firesmith consolida de forma bastante simples os requisitos de Segurança encontrados mais comumente, são eles[Firesmith 2003]:

•**Requisitos de Identificação:** Definem o grau e mecanismos que uma entidade utiliza para conhecer, ou reconhecer, entidades externas a ela como por ex: usuários, aplicações externas ou serviços, antes que estes possam interagir. Exemplos de Requisitos de Identificação:

- A aplicação deve identificar todas as suas aplicações clientes antes de permitir que elas tenham acesso a suas funcionalidades.
- A aplicação deve identificar todos os usuários antes de permitir que eles tenham acesso a suas funcionalidades.

•**Requisitos de Autenticação:** Definem o grau e mecanismo que uma entidade verifica, confirma ou nega, a identidade apresentada pelos externos, usuários, aplicações externas ou serviços, antes que estes possam interagir. Exemplos de Requisitos de Autenticação:

- A aplicação deve verificar a identidade de todos os seus usuários antes de permitir que eles tenham acesso a suas funcionalidades.
- A aplicação deve verificar a identidade de todos os seus usuários antes que estes possam alterar algum tipo de dado do sistema.

•**Requisitos de Autorização:** Definem o grau de acesso e privilégios que determinada(s) entidade(s) ou perfis receberá após ser autenticada e validada por alguma parte do sistema. Exemplos de Requisitos de Autorização:

- A aplicação deve permitir que cada usuário tenha acesso a todos os seus dados pessoais.
- A aplicação não poderá permitir que usuários tenham acesso às informações dos demais usuários.

•**Requisitos de Imunidade:** Define o grau em que uma entidade protege a si mesma de invasões, infecções ou alterações, por parte de programas maliciosos, por ex: Vírus, trojans, worms e scripts maliciosos.Exemplos de Requisitos de Imunidade:

- A aplicação deve conseguir se desinfetar de todo e qualquer arquivo, que seja detectado como danosos, se possível.

- A aplicação deve notificar o administrador de segurança e usuário logado, em caso de detecção de algum programa danoso, durante um processo de scan.

•**Requisitos de Integridade:** Define o grau no qual uma comunicação, ou dado, hardware e software, se protegem de tentativas de corrupção por parte de terceiros. Exemplos de Requisitos de Integridade:

- A aplicação deve prevenir a corrupção, não autorizada, de dados, mensagens e valores e qualquer outra entidade enviada pelo usuário.
- A aplicação deve garantir que os dados armazenados nela não estejam corrompidos.

•**Requisitos de Detecção de intrusão:** Define o grau no qual uma tentativa de acesso ou modificação não autorizado é detectado, registrado e notificado, pelo sistema. Exemplos de Requisitos de Detecção de intrusão:

- A aplicação deve detectar e registrar todas as tentativas de acesso que falharem nos requisitos identificação, autorização ou autenticação.
- A Aplicação deve notificar os responsáveis imediatamente sempre que algum perfil sem a correta permissão tente acessar uma área restrita mais de uma vez.

•**Requisitos de Não – Repudiação:** Define o grau no qual uma parte de um determinado sistema impede uma das partes das interações tomadas dentro, ou pelo sistema, por ex: uma troca de mensagens, neguem seu envolvimento em determinado momento.Exemplos de Requisitos de Não – Repudiação:

- A aplicação deve ser capaz de armazenar dados sobre as transações feitas por um usuário de modo a conseguir identificar com precisão qual usuário efetuou qual transação.
- A aplicação deve ser capaz de armazenar dados sobre as ações feitas por cada usuário de modo a conseguir identificar com precisão qual usuário efetuou qual ação.

•**Requisitos de Privacidade:** Também conhecido como Confidencialidade. Define o grau no qual dados importantes e comunicações são mantidos privados de acesso por parte de indivíduos ou programas. Exemplos de Requisitos de Privacidade:

- A aplicação deve garantir que usuários não possam acessar dados confidenciais de outros usuários.
- A aplicação deve garantir que toda e qualquer comunicação feita pelos usuários não poderá ser entendida em caso de captura.

•**Requisitos de Auditoria de Segurança:** Define o grau em que a equipe responsável pela segurança tem permissão para verificar o status e uso dos mecanismos de segurança

através da analise de eventos relacionados a estes. Exemplos de Requisitos de Auditoria de Segurança:

- A aplicação deve ser capaz de detectar e armazenar todas as transações feitas pelo usuário
- A aplicação deve prover um mecanismo através do qual o pessoal de administração consiga rastrear as ações de cada usuário dentro do sistema.

- **Requisitos de Tolerância a Falhas:** Define o grau em que uma entidade continua a executar sua missão, provendo os recursos essências a seus usuários mesmo na presença de algum tipo de ataque. Exemplos de Requisitos de Tolerância:

- A aplicação só pode apresentar um único ponto de falha.
- A aplicação não pode ter seu sistema de auditoria fora do ar.

- **Requisitos de Proteção Física:** Define o grau em que uma entidade se protege fisicamente de outra entidade. Exemplos de Requisitos de Proteção Física:

- Os dados armazenados em hardware devem ser protegidos contra danos físicos, destruição, roubo ou troca não autorizada.
- Aqueles que manuseiam os dados devem ser protegidos contra danos, morte e seqüestro.

- **Requisitos de Manutenção de Segurança de Sistemas:** Define o grau em que o sistema deve manter suas definições de segurança mesmo após modificações e atualizações.Exemplos de Manutenção de Segurança de Sistemas:

- A aplicação não deve violar seus requisitos de segurança após upgrade de dados, hardware ou componentes.
- A aplicação não deve violar seus requisitos de segurança após uma troca de dados, hardware ou componentes.

1.3. Relacionando Requisitos de Segurança e Padrões de Projeto

Nesse capítulo faremos uma divisão e classificação dos tipos de requisitos mencionados anteriormente e faremos a proposta do seu relacionamento com padrões de projeto, a fim de tentar garantir uma abordagem relacionada à implementação desses requisitos utilizando as especificações feitas pelo GoF[Gamma, 1996] de modo a tornar mais factível a implementação de requisitos de segurança.

A adoção desses padrões, seguindo a proposta neste trabalho, auxilia a construção do sistema garantindo que definições relacionadas aos requisitos de segurança possam ser mais bem estruturadas, alem de proporcionar uma maior facilidade relacionada às implementações das políticas de segurança mesmo que essas sejam definidas apenas em um segundo momento.

1.3.1 Divisão e Classificação

Uma vez definidos os tipos de requisitos de segurança e analisadas suas definições, iremos classificá-los de acordo seus propósitos, agrupando aqueles que mostrarem mais semelhança.

Inicialmente, tomaremos os requisitos de: Identificação, Autenticação, Autorização, Não Repudiação e Privacidade. Ao analisar esses requisitos podemos observar que todos tratam do mesmo assunto, *identificação de um ator*, seja ele usuário, sistema ou outra entidade que interage com o sistema em questão. Todos tratam da identidade que este ator possui. Respectivamente temos a Identificação do ator, a prova da Identificação, as permissões da identidade, a confirmação das ações da entidade e por fim os segredos ou sigilo da identidade. Todos esses requisitos giram em torno da criação da Identidade do individuo desde os primeiros passos, ou no caso de sistemas, telas, interações, casos de uso ou outros.

É possível visualizar também que, uma vez falado em criação de entidades relacionadas identidade, a arquitetura do sistema sofrerá impactos para que seja possível a adoção, e utilização, das identidades criadas. Indo além, podemos verificar que uma vez adaptada a arquitetura para a necessidade em questão, relacionada à criação da identidade, também será necessário adaptá-la para fazer uso dessa identidade.

Continuando a analise dos requisitos, podemos separar os requisitos de *Imunidade* e *Integridade*, como sendo dois requisitos que atingem diretamente a estrutura do sistema. Do ponto de vista deste, os requisitos de *Imunidade* visão garantir que o sistema esteja imune a contaminações por partes dos atores e os requisitos de *Integridade* visão garantir que a estrutura do sistema proveja um mecanismo integro para a comunicação entre esses atores. Dessa forma ambos os requisitos tem uma relação direta com a estrutura do sistema uma vez que será necessário alterar a estrutura do sistema de modo a adotar soluções para esses requisitos.

Seguindo, temos os requisitos de *Detecção de Intrusão, Auditoria de Segurança* e *Tolerância a Falhas*, que tratam das questões relacionadas às ações tomadas junto ao sistema. No primeiro caso, *Detecção*, temos um requisito que trabalha de forma preventiva, que visava disponibilizar mecanismo para detecção e notificação em caso de acesso indevido; já a auditoria vem como um mecanismo para trabalhar questões de forma mais reativa, ou seja atitudes que podem ser tomadas a partir da comprovação e constatação de ações, um requisito de auditoria deve contemplar o registro de ações como também mecanismos para futuras consultas[Schumacher, 2006], diferente de *Tolerância a falhas* que além de reativo, define qual o comportamento o sistema terá em caso de falha, também é preventivo ao garantir que falhas em entidades do sistema não comprometam o restante do sistema. Por tanto, os requisitos mencionados trabalham a questão dos comportamentos tomados dentro do sistema.

Por fim analisando os requisitos de Manutenção de Segurança do Sistema e de Proteção Física temos, nesse, um requisito que trata mais da questão física, como o próprio nome remete, aonde a preocupação vai além do âmbito software, por tanto fora

do nosso escopo de analise. Já o requisito de Manutenção tem um comportamento horizontal em relação aos demais requisitos, visto que esse trata da manutenção das demais necessidades do sistema relacionadas à segurança, indiretamente ele trata dos requisitos responsáveis por tais necessidades. Aparece por tanto um requisito não considerável do ponto de vista de software e um que é a soma de demais requisitos.

Organizando-os de acordo com suas características temos:

1-Requisitos de Identificação, Autenticação, Autorização, Não-Repudiação e Privacidade sendo relacionados com criação.

2-Requisitos de Imunidade e Integridade relacionados com a estrutura do sistema.

3-Requisitos de Detecção de Intrusão, Auditoria e Tolerância a Falhas relacionados com comportamentos dos atores no sistema.

4-Requisitos de Manutenção de Segurança de Sistemas relacionado com os demais requisitos.

Sob essa óptica e utilizando umas das classificações do GoF, podemos separa os requisitos de acordo com seus propósitos. A partir das características apresentadas, vamos separá-los em 3 grupos de acordo com esse critério propósitos, são eles, Criação, Estrutural e Comportamental. A tabela 1 mostra essa divisão:

Como mencionado anteriormente, Os requisitos de Proteção física, por tratarem de questões físicas relacionadas ao ambiente físico do sistema não são contemplados dentro dessa estrutura.

Propósito		
Criação	**Estrutural**	**Comportamental**
Requisitos de Identificação	Requisitos de Imunidade	Requisitos de Detecção de intrusão
Requisitos de Autenticação	Requisitos de Integridade	Requisitos de Auditoria de Segurança
Requisitos de Autorização		Requisitos de Tolerância a Falhas.
Requisitos de Não–Repudiação		
Requisitos de Privacidade		
Requisitos de Manutenção de Segurança de Sistemas		

Tabela 1: Divisão de requisitos de acordo com os tipos de seus propósitos.

1.3.2 Relação com Padrões de Projeto

Conhecendo agora a divisão da sessão anterior, o próximo passo é tentar tratar cada um desses requisitos, de acordo com seus propósitos, de forma a tentar garantir uma relação com padrões de projeto.

É importante lembrar que diferente dos *Security Patterns*[Schumacher, 2006], que se apresentam em sua maioria como Padrões Arquiteturais e da abordagem proposta por Weis, Michael[Weiss 2008] estamos abordando o uso de Padrões de Projeto na construção de arquiteturas com o intuito de tornar a adoção dos requisitos de segurança, desses sistemas, mais simples e próxima aos desenvolvedores, que, como mencionado anteriormente, tem pouco ou nenhum conhecimento do ponto de vista de segurança de software, alem disso podemos também diminuir os impactos de implementações tardias das políticas de segurança uma vez que muitas vezes estas não estão claras no inicio da criação dos sistemas.

Como vimos, o GoF[Gamma, 1996] classifica seu padrões por dois critérios, sendo o critério de propósito responsável por classificar os padrões de acordo com o que o padrão faz. Dessa forma existem padrões responsáveis pela criação de entidades, estrutura do sistema e comportamento das entidades, respectivamente: Padrões de Criação, Estrutura e Comportamento. Sendo assim é possível ver um relacionamento direcionado entre tipos de requisitos e padrões. Figura. 2

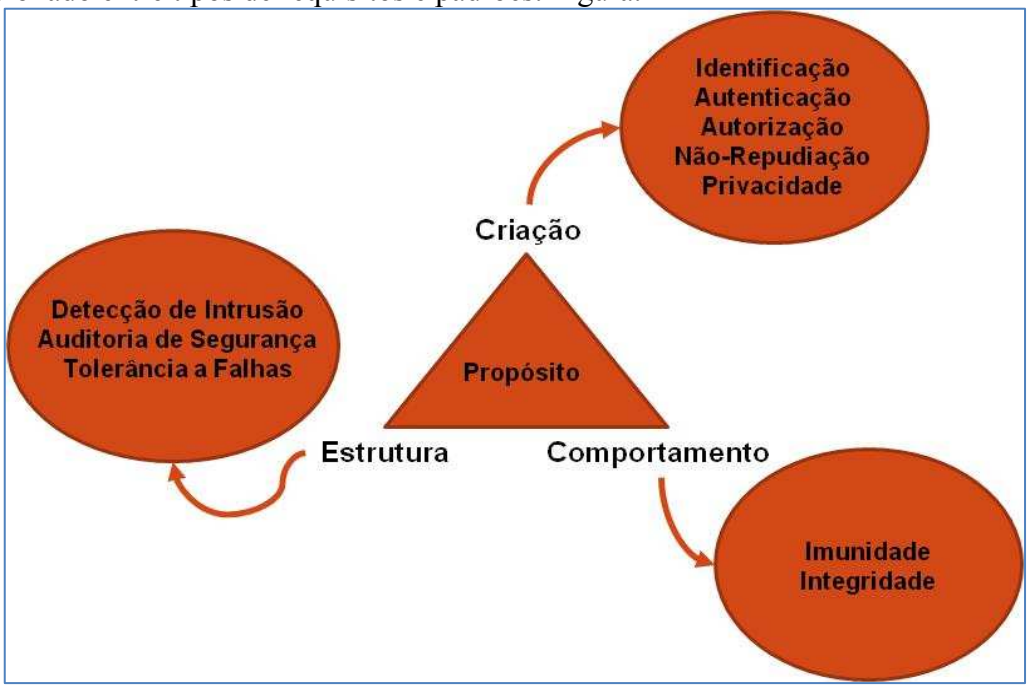

Figura 2: Divisão dos requisitos.

Uma vez divididos os tipos de requisitos e os relacionando com padrões de projeto, resta agora entender como utilizá-los de modo a auxiliar a criação das arquiteturas a fim de diminuir os impactos.

1.3.3. Uso

Por fim, uma vez entendida a relação entre os requisitos e seus respectivos padrões de projeto iremos fazer uso dessa relação, para trabalhar a situação mencionada no capítulo 3. Vamos propor uma abordagem utilizando essencialmente OO e padrões de projeto para tentar montar uma solução de software que esteja preparada para receber implementações de segurança mesmo que estas não tenham sido completamente definidas durante etapas de iniciais do projeto. Respaldado na afirmação de Yoder que mostra que padrões de projeto auxiliam na construção de arquiteturas[Yoder, 1996] utilizaremos a relação criada na sessão anterior para, através da adoção dos padrões, trabalhar problemas de segurança. Nesse situação é importante que foquemos no uso dos padrões de forma correta e não no retorno ou comportamentos do padrão, por tanto sempre recomendaremos o uso de implementações *stubs* dos padrões em questão, para que esses possam ser adicionados ao código e criando o mínimo impacto nas execuções iniciais do sistema e diminuindo impactos de modificações futuras.

Importante salientar que estamos sugerindo uma abordagem em código para um problema conhecido arquitetural, dessa forma estamos trabalhando com as soluções apresentadas pelo GoF para resolver problemas relacionados a requisitos de segurança de software e não hardware.

Para os relacionamentos entre requisitos e padrões mostrados anteriormente, teremos situações relacionadas à Criação, Estrutura e Comportamento do sistema, tais podem ser abordados como mostrado nas sessões seguintes.

Por questões meramente de visualização utilizaremos exemplos usando a linguagem Java, desde já deixamos claro que não será utilizada nenhuma API específica na nossa abordagem, mesmo que em um dos estudos de caso apresentados utilize tecnologias especificas dessa plataforma.

1.3.4 Criação.

Os requisitos relacionados à criação devem ser tratados através da adoção de algum dos padrões criacionais do GoF.

Requisitos de Identificação, Autenticação e Autorização tratam da criação da(s) entidade(s) responsável(is), respectivamente, pela identificação do individuo ou sistema, da validação do mesmo e das permissões que este possui. Estes por si só utilizarão um padrão de criação que será responsável por criar as tais entidades, e se for o caso, com seus métodos vazios caso maiores informações relacionadas ao requisito não sejam informadas.

Tomemos como exemplo um *Factory Method*[Gamma 1996] ou *Abstract Factory*[Gamma 1994] na Figura 3.

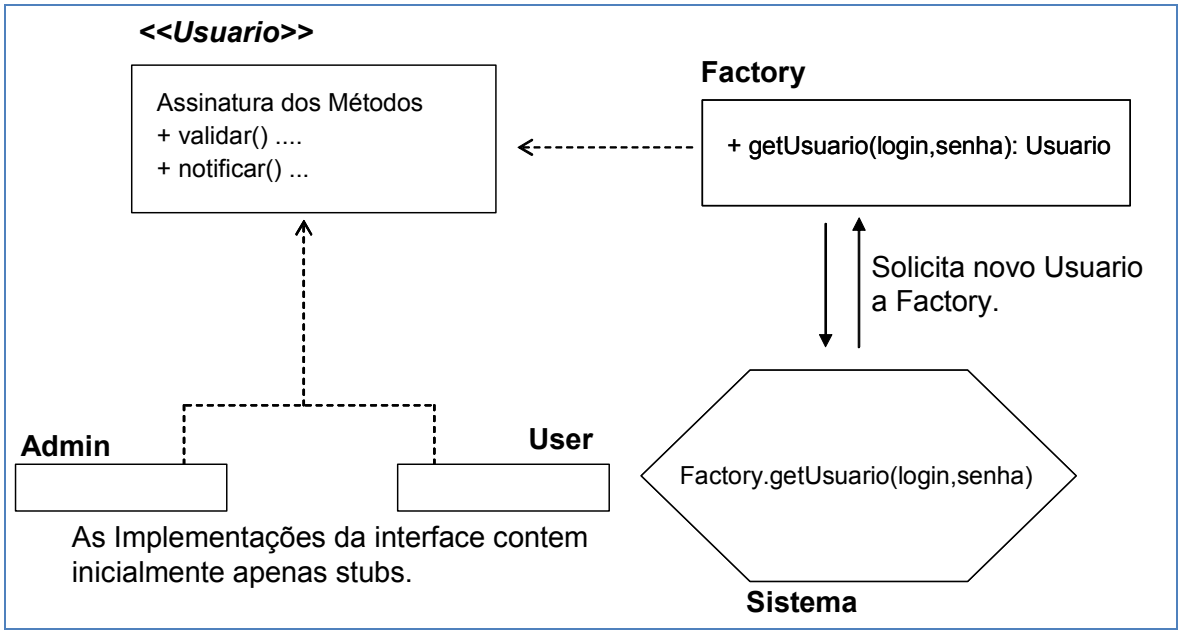

Figura 3: Esquema de criação da entidade Usuário utilizando Factory

A Factory em questão contemplará:

- A definição de uma interface
- A criação de uma classe para retornar as instâncias das implementações dessa interface.
- A criação de implementações da interface definida.

A *Factory* em questão, deve estar apta a retornar as implementações definidas sempre que for requisitada pelo sistema, no exemplo a cima adotamos 2 entidades retornáveis pela *Factory*, são elas *Admin* e *User*, que representam usuários do sistema do tipo Administrador e Usuário, ambos com permissões a serem definidas. A interface comum a essas duas implementações é chamada *Usuário* e possui métodos relacionados à validação das permissões dessa entidade.

O sistema deverá fazer uso dessa, ou de demais *Factorys* sempre que for necessária a criação das entidades relacionadas aos atores do sistema ou sempre que se julgar necessária a criação de uma nova entidade relacionada a algum requisito de Criação. Nesse momento o *Factory* retornará uma instância do tipo *Usuario* que, no exemplo, poderá ser tanto *Admin* quanto *User* e estes por sua vez contem a implementação da interface *Usuário*, tendo seus métodos implementados na forma de *stubs*, em outras palavras, os métodos estarão vazios ou retornando valores positivos, por exemplo *true* para valores *boleanos*. Tomemos a Figura 2 para ilustrar a implementação *stub* da classe *Admin*.

```
public class Admin implements Usuario {

        public boolean validar() {
            // TODO: Modificar esse bloco, acrecentando
            // a correta validação do Usuario
            return true;
        }

        public boolean temPermissao(int id) {
            // TODO: Modificar esse bloco, acrecentando
            // a correta verificação das permissões do Usuario
            return true;
        }

}
```

Figura 4: Exemplo de Stub.

Na figura 4, podemos ver os métodos *validar()* e *temPermissao()* retornando o valor *true* para qualquer caso. Dessa forma mesmo que o foco do desenvolvedor seja, inicialmente, validar algum requisito relacionado a algum requisito funcional do sistema, ao utilizar essa simples estrutura podemos garantir que mesmo o foco do desenvolvedor sendo outro, essa adoção não trará impactos nesse momento inicial e os reduzirá em caso de mudanças no futuro quando as reais especificações e validações relacionadas a políticas de segurança serão definidas.

Dentro ainda dos Requisitos ligados a Criação, temos ainda os requisitos de Não–Repudiação e de Privacidade que devem utilizar os objetos criados anteriormente pela *Factory* em questão. Dessa forma temos como garantir que, para o requisito de Não-Repudiação, as atividades, ou ações, executadas pelos usuários do sistema não possam ser negadas desde que sejam associadas às identidades criadas e disponibilizadas pela *Factory*, ou seja, ao se aplicar também critérios de auditoria temos um mecanismo de identificação que será utilizado para criar relação entre Ator-Ação onde o ator é, inicialmente, um *stub* a ser modificado posteriormente e a ação será também um *stub*, como veremos a seguir.

Para garantir e adotar o requisito de Privacidade teremos um mecanismo onde utilizaremos as entidades criadas, associadas as suas permissões, utilizando-as de modo a garantir que apenas o usuário com a correta permissão para tal, possa interagir com alguma informação, dado ou até mesmo área do sistema.

É importante mostrar que estamos trabalhando inicialmente com o uso de uma Factory, mas apenas como uma sugestão, no estudo de caso veremos que um padrão relacionado ao propósito de comportamento(*command*), juntamente com um criacional(*singleton*) serão utilizados para implementar requisitos de Autenticação e Autorização.

1.3.5 Estrutural.

Os requisitos relacionados à estrutura do sistema serão classificados nessa categoria. São eles: Requisitos de Imunidade e Requisitos de Integridade.

Mais do que tratar de como a estrutura do sistema será montada, ou como classes poderão ser adaptadas para outros usos[Gamma, 1996], essa categoria também se preocupa em montar o sistema de forma a facilitar o relacionamento entre as entidades através do uso de informações presentes no sistema como base para decisão de ações a serem tomadas.

Tomemos como exemplo o padrão *Adapter,* esse padrão tem como principal característica permitir modificações posteriores nas entidades do sistema, garantindo o mínimo de mudanças estruturais. Ele provê um mecanismo para transformar uma entidade em outra através da adaptação das chamadas de seus métodos[Gamma 1996].

No nosso exemplo, os objetos que transitarem pelo sistema serão fruto das criações de identidade e permissões dos *Usuários*. Uma vez criado os objetos, poderemos alterá-los para retornar valores ou ter comportamentos diferentes através da adoção do *Adpater*. Nesse cenário a interface definida anteriormente não será modificada, mas sim utilizada para checar as permissões e validações, tais comportamentos por sua vez serão utilizados pelo *Adapter* que implementará a interface e internamente repassará os comportamentos para as entidades adequadas, em outras palavras, o adaptador será o ponto de entrada para diferentes verificações a partir da interface e objetos conhecidos. Esses novos objetos deverão ser utilizados para garantir os requisitos de Imunidade e Integridade.

No exemplo iniciado na sessão anterior, queremos que o requisito de Imunidade garanta que apenas usuários, ou entidades, com permissão adequada possam ter acesso ao ambiente do sistema, ou permissão para escrita, leitura, ou qualquer outro tipo de ação, isso será alcançado através da checagem do objeto que representa o usuário e suas permissões. Futuramente ao se modificar as entidades para que essas possam executar diferentes tipos de validações e checagens, ou até mesmo executar diferentes procedimentos quando seus métodos conhecidos forem chamados, utilizaremos um Adaptador para que as modificações imposta a entidade sejam mínimas e não alterem as assinaturas dos métodos, dessa forma temos que o adaptador ficará responsável por receber as chamadas do sistema e de encaminhá-las a entidade correta. Figura 5 ilustra a idéia do Adaptador.

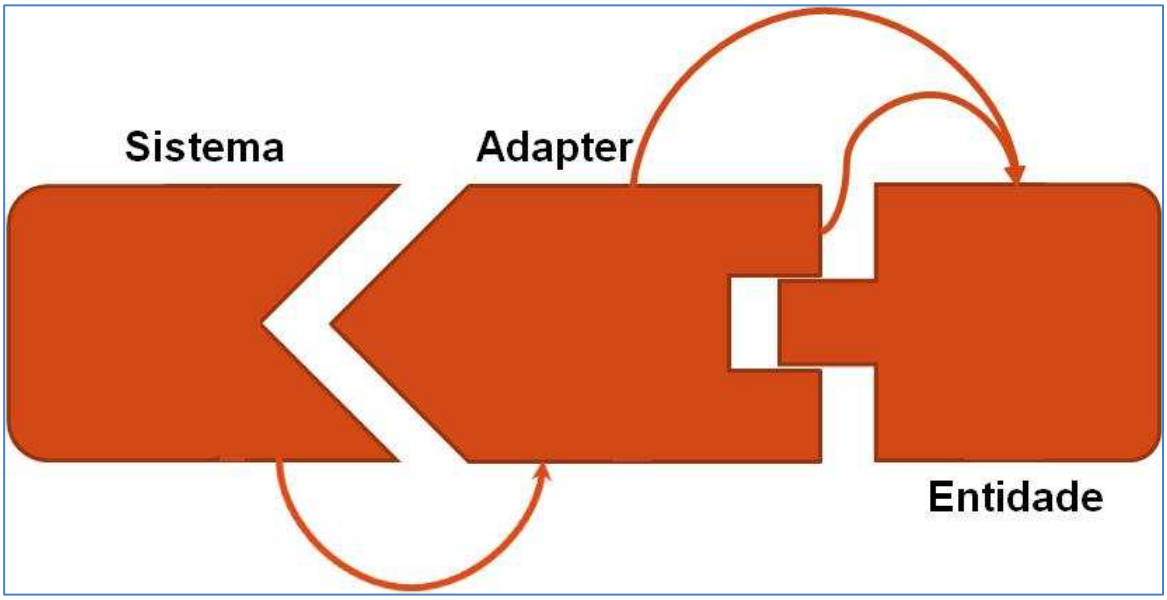

Figura 5: Ilustração do adaptador.

Sendo assim, será criada uma nova estrutura com as mesmas características da entidade adaptada.

O novo objeto deve ser checado sempre que acessos a algum dado ou hardware sejam efetuados, para que se possa garantir também o requisito de Integridade. Nesse cenário trabalharemos com uma checagem de permissão para que apenas entidades autorizadas possam executar determinadas alterações. Figura 6.

```
if(user.temPermissao(Usuario.PERMISSAO_ESCRITA)) {
                    //Executar restante do código
}
```

Figura 6: Checagens do Usuário.

Ainda nesse cenário, pode-se estender esse tipo de abordagem para acrescentar às funcionalidades de verificação, algoritmos para garantir a integridade também do conteúdo dos dados acessados. Nessa, poderemos fazer uso do adaptador para modificar posteriormente tais verificações modificando os algoritmos para melhor atender as necessidades do sistema.

1.3.6 Comportamental.

Os requisitos relacionados aos comportamentos do sistema são classificados nessa categoria. Requisitos de Detecção de intrusão, Auditoria de Segurança e Tolerância a Falhas tratam da notificação do sistema em caso de determinados eventos ocorrerem, bem como da atitude tomada pelo sistema quando tais eventos ocorrem. No primeiro

caso eventos de falha relacionados a acessos indevidos, tanto a sua detecção quanto ação quando dos acontecimentos. No segundo há uma gama maior de possibilidades que devem ser primeiro detectados e em seguida armazenados para futura consulta. Por fim temos ações tomadas para a prevenção de falhas junto ao sistema.

Para esse tipo de situação propõe-se a adoção dos padrões relacionados ao Comportamento. Por ex, *Observer*.

Teremos uma entidade responsável, o *Observer*, por receber mensagens, relacionadas a determinados eventos. Para eventos relacionados à intrusão deve-se avisar ao *Observer* para que esse tome alguma ação desde notificar os responsáveis até mesmo negar o acesso, para eventos relacionados à auditoria devemos notificar o responsável para que esse armazene os eventos enviados de modo a possibilitar uma futura consulta e por fim em caso de eventos provindos da detecção de falhas avisarem ao *Observer* para que esse tome alguma atitude a fim de tornar transparente ao usuário do sistema que tal falha aconteceu. A Figura 7 mostra uma abordagem geral para o caso do *Observer*.

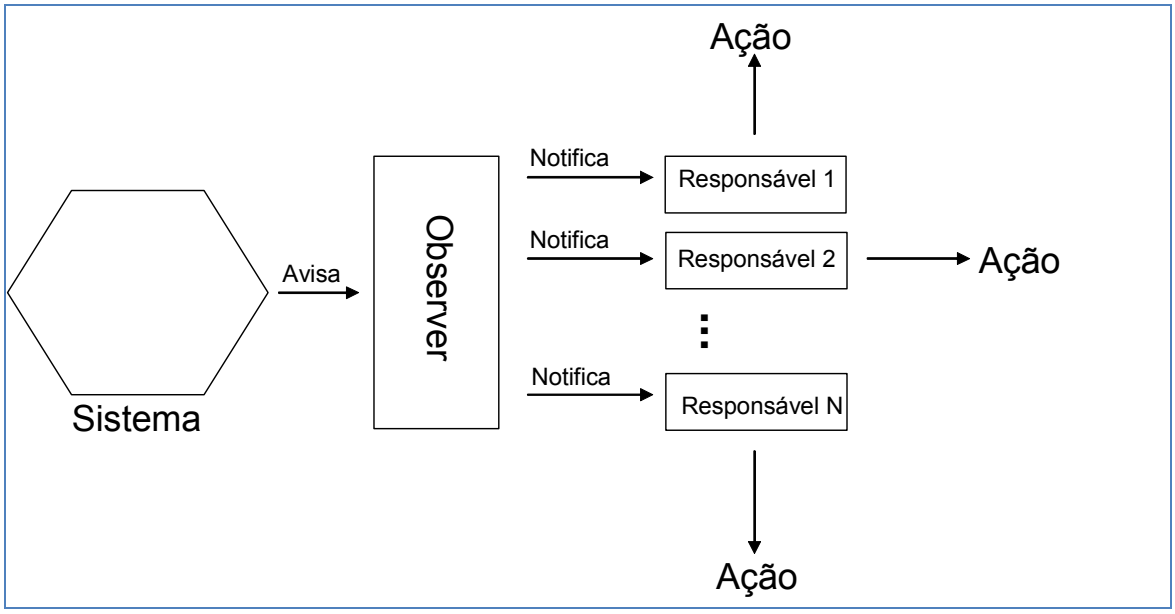

Figura 7: Notificações ao Observer

A partir dessas notificações o *Observer* deverá notificar todas as entidades nele registradas. Todas as entidades que desejarem ser notificadas em caso de algum tipo ação relacionadas à Auditoria, Detecção de Intrusão ou Tolerância a Falhas deverão implementar uma interface especifica e se registrar no *Observer* correto.

Uma vez registradas as entidades junto ao *Observer* e este recebendo a notificação de alguma ação as entidades deverão ser notificadas a fim de efetuar alguma ação sejam elas bloquear acesso, restringir uso, reiniciar sistema ou mesmo apenas notificar os responsáveis. Inicialmente tal implementação da interface deverá ser vazia de modo a que essencialmente nenhuma ação seja tomada, quando as políticas relacionadas a tais requisitos forem determinadas a estrutura para tomada de ações já estará montada.

1.3.7 Requisito de Manutenção de Segurança de Sistemas.

Por fim tratamos do requisito mais abrangente. Pela definição de Firesmith[Firesmith 2003][Firesmith 2004], vemos que esse requisito cuida do sistema após a instalação de uma nova versão, garantindo que antigas definições de segurança não sejam perdidas. Através da padronização do código do sistema pelo uso dos padrões referentes à aos requisitos de Criação, Comportamento e Estrutura, teremos a possibilidade de uma atualização mais fácil, inicialmente os *stubs* serão criados e aplicados, se em algum momento modificações relacionadas ao escopo de segurança forem necessárias é possível fazê-las apenas preenchendo os métodos vazios. Tal procedimento tende, nesse momento, a ser mais fácil e menos custoso uma vez que os locais passíveis de alteração já são conhecidos, bem como os impactos de suas chamadas.

Espera-se que as atualizações dos requisitos de segurança, sejam efetuadas de forma menos danosa uma vez que agora adotam um padronização em seu código através da utilização dos padrões na definição e construção de sua arquitetura.

1.3.8 Consolidação

A Tabela 2 mostra de forma consolidada a relação criada entre requisitos, padrões sugeridos e sua utilização.

	Requisitos	Padrões	Uso
Criação	Identificação Autorização Autenticação Não Repudiação Privacidade	Factory Method Or Abstract Factory	Criar interfaces e implementações retornáveis pelo Factory sempre que alguma entidade responsável por identificação for criada.
Estrutura	Imunidade Integridade	Adapter	Faça com que as diversas áreas do sistema chequem as identidades criadas anteriormente para validar permisoes. Utilize o Adapter para mudar essas entidades.
Comportamento	Detecção de Intrusão Auditoria Tolerância a Falhas	Observer	Faça com que os responsáveis pelas ações registrem-se junto ao Observer para que este notifique-as em casos específicos. Faça com que essas entidades estejam vázias.

Manutenção de requisitos de segurança

Tabela 2: Resumo das Atribuições.

A intenção desse trabalho foi mostrar que existe uma maneira fácil e ágil para se implementar requisitos de segurança. Como foi levantado, existe uma grande negligência no que diz respeito a esse aspecto em particular, muitas vez por falta de detalhes e outras por desconhecimento, espera-se que através da implementação desses padrões as futuras, implementações muitas vezes na fase final do desenvolvimento, possam ser feita garantindo um mínimo de qualidade e aceitação para o produto sem um grande impacto de cronograma. Para tal, inicialmente analisamos a classificação proposta pelo GoF, em seguida analisamos e entendemos os tipos de requisitos de segurança, posteriormente classificamos tais requisitos com base em uma relação com padrões de projeto e por fim mostramos o como utilizá-los sem prejudicar o projeto, criando uma estrutura baseada nos requisitos deste para garantir segurança da informação dentro do contexto de software.

Os pontos abordados nesse trabalho seguem apenas como uma sugestão inicial no que diz respeito a que padrões utilizarem para cada requisito. Cabe aos arquitetos, analistas, desenvolvedores avaliarem melhores os Padrões de Projetos apresentado pelo GoF, adaptando-os de forma semelhante a apresentada no trabalho, adequando dessa forma sua realidade a uma arquitetura mais apropriada, atendendo os requisitos de segurança que forem necessários.

Referências

7498-2, ISO. 1989. *Information processing systems -- Open Systems Interconnection -- Basic Reference Model -- Part 2: Security Architecture*.

Adikari, Jithra. 2006. "Efficient Non-Repudiation for Techno-Information Environment." *First International Conference on Industrial and Information Systems* 454-458. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4216631.

Devanbu, Premkumar T, and Stuart Stubblebine. 2000. "Software Engineering for Security: a Roadmap. "pp. 227-239 in *The future of Software Engineering*. ACM Press.

Dhillon, Gurpreet. n.d. *Principles of Information Systems Security: Texts and Cases*. 1 edition. Wiley.

Firesmith, D.G. 2004. "Analyzing and Specifying Reusable Security Requirements. "in *Eleventh International IEEE Conference on Requirements Engineering (RE'2003) Requirements for High-Availability Systems (RHAS'03) Workshop*. Citeseer

Firesmith, D.G. 2003. "Engineering security requirements." *Journal of Object Technology* 2:53–68.

Gamma, E, R Helm, R Johnson, and J Vlissides 1994. n.d. *Design Patterns: Elements of Reusable Object-Oriented*. 1st edition. Addison-Wesley Professional

Haley, Charles B, Jonathan D Moffett, Robin Laney, and Bashar Nuseibeh. 2006. "A framework for security requirements engineering. "pp. 35-42 in *SESS '06: Proceedings of the 2006 international workshop on Software engineering for secure systems*. New York, NY, USA: ACM Press http://dx.doi.org/10.1145/1137627.1137634.

Haley, Charles B, Robin C Laney, and Bashar Nuseibeh. 2004. "Deriving security requirements from crosscutting threat descriptions. "pp. 112-121 in *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*. New York, NY, USA: ACM Press http://dx.doi.org/10.1145/976270.976285.

Information Technology - Security Techniques - Evaluation Criteria for IT Security. Part 1: In. Geneva Switzerland: ISO/IEC Information Technology Task Force (ITTF).

Khomh, F, and Y G Gueheneuc. 2008. "Do Design Patterns Impact Software Quality Positively?. "pp. 274-278 in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*. http://dx.doi.org/10.1109/CSMR.2008.4493325.

Kunyu, Peng. 2009. "AN IDENTITY AUTHENTICATION SYSTEM BASED." *Identity*.

Lutz, R.R. 2000. "Software engineering for safety: a roadmap. "p. 213–226 in *Proceedings of the Conference on The Future of Software Engineering*. ACM http://portal.acm.org/citation.cfm?id=336512.336556.

Pawlak, Piotr, Bartosz Sakowicz, Piotr Mazur, and Andrzej Napieralski. 2009. "Social Network Application based on Google Web." *Source* 461-464.

Peiris, Hasala, Lakshan Soysa, and Rohana Palliyaguru. 2008. "Non-Repudiation Framework for E-Government Applications." *2008 4th International Conference on Information and Automation for Sustainability* 307-313. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4783950.

Samarati, Pierangela, and Sabrina De Capitani di Vimercati. 2000. "Access Control: Policies, Models, and Mechanisms. "pp. 137-196 in *FOSAD*, vol. 2171, *Lecture Notes in Computer Science*, Riccardo Focardi and Roberto Gorrieri. Springer.

Schumacher, Markus, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. 2006. *Security Patterns : Integrating Security and Systems Engineering (Wiley Software Patterns Series)*. John Wiley & Sons http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0470858842.

Sindre, Guttorm, and Andreas L. Opdahl. 2004. "Eliciting security requirements with misuse cases." *Requirements Engineering* 10:34-44. http://www.springerlink.com/index/10.1007/s00766-004-0194-4.

Steiner, Peter. 1993. "On the internet nobody knows you´re a Dog." *The New Yorker* 69:61.

Wang, Zhi-Hui, Ming-Chu Li, Mao-Hua Chen, and Chin-Chen Chang. 2009. "A New Intelligent Authorization Agent Model in Grid." *2009 Ninth International Conference on Hybrid Intelligent Systems* 394-398. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5254309.

Weiss, Michael, and Haralambos Mouratidis. 2008. "Selecting Security Patterns that Fulfill Security Requirements." *2008 16th IEEE International Requirements Engineering Conference* 169-172. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4685666.

Yoder, Joseph, and Jeffrey Barcalow. n.d. "Architectural patterns for enabling application security." *Urbana* 51:61801. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.3950&rep=rep1&type=pdf.

Zerkle, Dan, and Karl Levitt. 1996. "NetKuang -- A Multi-Host Configuration Vulnerability Checker. "in *in Proceedings of the 6th USENIX Unix Security Symposium*.