UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURÍ
Bacharelado em Sistemas de Informação
Pedro Galiciolli Orlando

PADRÕES DE PROJETO DE SEGURANÇA DA INFORMAÇÃO: um estudo sobre os padrões mais relevantes no cenário da engenharia de software

Diamantina, MG
2019

Pedro Galiciolli Orlando

PADRÕES DE PROJETO DE SEGURANÇA DA INFORMAÇÃO: um estudo sobre os padrões mais relevantes no cenário da engenharia de software

Trabalho de conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação como parte dos requisitos exigidos para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Eduardo Pelli

Diamantina, MG
2019

Pedro Galiciolli Orlando
Padrões de projeto de segurança da informação/ Pedro Galiciolli Orlando. – Diamantina, MG, 2019-
115 p. :
Orientador: Prof. Dr. Eduardo Pelli

Trabalho de Conclusão de Curso –
Universidade Federal dos Vales do Jequitinhonha e Mucurí, 2019.

1. Padrões de Projeto. 2. Segurança da Informação. 3. Disponibilidade. 4. Integridade. 5. Confidencialidade

Pedro Galiciolli Orlando

PADRÕES DE PROJETO DE SEGURANÇA DA INFORMAÇÃO: um estudo sobre os padrões mais relevantes no cenário da engenharia de software

Trabalho de conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação como parte dos requisitos exigidos para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Eduardo Pelli

Aprovado em: 19/07/2019.

Prof. Dr. Eduardo Pelli
Orientador

Prof. Dr. Alexandre Ramos Fonseca
UFVJM - Universidade Federal dos Vales do Jequitinhonha e Mucuri

Prof. M.e. Áthila Rocha Trindade
UFVJM - Universidade Federal dos Vales do Jequitinhonha e Mucuri

Diamantina, MG

“Nobody exists on purpose,
nobody belongs anywhere,
everybody’s gonna die.
Come watch TV.
(Rick & Morty, Temporada 1, Episódio 8)

RESUMO

Praticamente todos os processos realizados atualmente estão relacionados, de alguma forma, a sistemas de informação. Com isso, aumentam também as diversas possibilidades de ataque às informações contidas nesses sistemas. Padrões de projeto são soluções consolidadas para problemas recorrentes em relação ao desenvolvimento de software. Os Padrões de Projeto voltados à segurança da informação, no entanto, não possuem um método específico de categorização, o que dificulta o processo de escolha de qual padrão utilizar para cada situação, o que resulta em softwares menos seguros e dados menos protegidos contra invasores. O objetivo deste trabalho foi, além de analisar o cenário dos padrões de projeto de softwares voltados à segurança da informação, escolher os mais relevantes e apresentá-los de modo a facilitar a busca e a escolha por esses padrões. Através de extensa revisão e filtragem da literatura, foram selecionados três pares de padrões de projeto, cada par representando um pilar da segurança da informação: *Replicated* e *Checkpointed System* representando a disponibilidade, *Input Validator* e *Secure Access Layer* representando a integridade e, por fim, *Authenticator* e *Authorization* representando a confidencialidade. Os padrões foram apresentados e implementados e, por fim, foi realizada uma análise desses padrões, comparando-os e expondo suas vantagens e desvantagens.

Palavras-chave: Engenharia de Software. Segurança da Informação. Confidencialidade. Integridade. Disponibilidade

ABSTRACT

Pretty much every process we do nowadays is related, in some level, to information systems. With that, the possibilities for an attack to happen to any of these systems also increase. Design patterns are consolidated solutions to recurring problems related to software development. Security design patterns, though, don't have a specific categorization method, wich hampers the choosing proccess of which pattern to use in each situation, which results in less secure software and data less protected against invasors. This work's goal was, besides assessing secure design patterns' landscape, to choose the patterns that are most relevant and present them in order to ease searching and choosing these patterns. Through extense literature filtering and review, three pairs of patterns were picked, each one representing one of information security's pilars: Replicated and Checkpointed System for availability, Input Validator and Secure Access Layer for integrity and, finally, Authenticator and Authorization for confidentiality. Said patterns were presented and implemented and, in the end, an more was made of those patterns, comparing them and poiting its pros and cons.

Keywords: Software Enfineering. Information Security. Confidentiality. Availability. Integrity.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GoF	Gang of Four
POO	Programação Orientada a Objetos
WMP	Workload Management Proxy

SUMÁRIO

1 INTRODUÇÃO

A segurança da informação nunca esteve tão em pauta como atualmente. Frequentemente descobre-se vazamentos e invasões a sistemas dos mais variados tipos, resultantes dos mais diversos métodos de ataques. Em um cenário em que nem políticos estão isentos de terem seus dados mais sigilosos acessados e divulgados, é cada vez mais grave negligenciar aspectos de segurança de aplicações. Ataques como o *Stuxnet* ou como as brechas de segurança da *Play Station Network* causaram tanto perdas materiais, quanto atrasos na economia e no desenvolvimento dos países afetados em anos (DANGLER, 2013).

A segurança da informação é sustentada por três pilares: confidencialidade, que significa que só quem tem acesso àquela informação é capaz acessá-la; integridade, que significa que as informações armazenadas são íntegras e confiáveis e; disponibilidade, que significa que as informações são acessíveis (a quem é permitido o acesso) a qualquer momento.

Durante as fazes de desenvolvimento de um software, diversos problemas podem aparecer. Para vários deles, já existem mecanismos de solução testados e comprovados chamados padrões de projeto. A existência desses padrões torna a fabricação de um software menos custosa e mais segura.

Por outro lado, o grande número de informações e ferramentas disponíveis para lidar com a proteção dos sistemas torna difícil uma estruturação concisa e padronizada dessas informações. Uma das ferramentas que apresenta esse déficit de padronização são os padrões de projeto para o desenvolvimento de software, voltados à segurança da informação. Subcategoria dos padrões de projeto, os padrões de projeto de segurança da informação não apresentam uma organização oficial dos seus elementos. Essa falta de organização acarreta em uma maior dificuldade em escolher qual tipo de padrão de projeto deve ser utilizado para solucionar determinado problema. Röser (2012) defende que nem mesmo as mais precisas categorizações fornecem ajuda o suficiente para quem procura descobrir qual padrão utilizar. Ainda segundo os autores, uma boa categorização permite ao usuário identificar um conjunto de padrões aplicáveis a determinado problema, mas as categorizações existentes não são capazes de guiar um usuário nessa decisão.

Por sua vez, a dificuldade dessa escolha acarreta em mais tempo gasto na projeção de um software e, consequentemente, mais custo. A falta de padronização inicia um efeito dominó que, muitas vezes, termina em desenvolvedores que tomam a decisão de diminuir o tempo e recursos na fabricação de um software em detrimento de mecanismos de segurança adequados. Hafiz, Adamczyk e Johnson (2007) afirma que, por mais que construir sistemas seguros seja difícil, adaptar, retroativamente, mecanismos de segurança a um sistema já existente é ainda mais difícil.

Em última análise, a falta de organização das ferramentas disponíveis para desenvolvedores dá origem a sistemas menos seguros que comprometem dados, dinheiro, reputação e qualquer outro fator que esteja relacionado à falta de privacidade na vida de uma pessoa.

1.1 Objetivos Específicos

Posto isso, propôs-se, com este trabalho, dois pontos principais: em um primeiro momento, realizar uma análise do cenário dos padrões de projeto de segurança e seus métodos de classificação para que, em seguida, fosse feita a apresentação dos padrões mais relevantes, selecionados através dessa análise, de modo a fornecer um guia conciso e confiável para a utilização dos mesmos.

1.2 Estrutura do Texto

O presente trabalho está estruturado da seguinte maneira: o Capítulo 1 é a introdução; o Capítulo 2 apresenta a revisão de literatura realizada, assim como seus resultados; o Capítulo 3 explica a metodologia utilizada para este trabalho; o Capíutlo 4 apresenta os resultados da pesquisa e; por fim, o Capítulo 5 trata-se das conclusões do trabalho.

2 REFERENCIAL TEÓRICO

Para entender melhor o escopo desse trabalho, é necessário compreender alguns conceitos: Programação Orientada a Objetos, Segurança da Informação e Padrões de Projeto.

2.1 Programação Orientada a Objetos

O Paradigma Orientado a Objetos tem como objetivo uma representação do mundo real e se baseia em 5 pilares: Abstração, Encapsulamento, Herança, Polimorfismo e Composição. Através de classes, que representam as entidades do mundo real, e objetos, que são as instanciações dessas classes, a programação orientada a objetos (ou POO) apresenta vantagens importantes para o desenvolvedor e utilizador do sistema, que se torna flexível e, ao mesmo tempo, robusto. A flexibilidade permite, ao desenvolvedor, mais facilidade de realizar alterações e ajustes no código, ao passo que a robustez significa que o software será confiável e realizará as atividades que se propõe a realizar de forma correta e precisa.

2.1.1 Abstração

Como o objetivo da POO é representar o mundo real, é necessário definir o que cada objeto poderá realizar dentro do sistema. Segundo Richard Wiener (2000), a abstração de dados associa um tipo de dado com um conjunto de operações que podem ser realizadas por aquele tipo de dado. Para essa abstração, três pontos são levados em conta:

Identidade

O objeto criado deve ter uma identidade que permita que ele seja identificado no sistema. Para que não haja conflitos, cada instanciação da classe Carro, por exemplo, deve ter um nome diferente, e cada um será único em relação aos demais, por mais que apresente as mesmas características. Uma classe consiste em uma coleção de objetos, enquanto cada objeto é uma instância de uma classe (MENDES, 2010).

Propriedades

Em POO, as características de um objeto (que, no caso do carro, podem ser a cor, ano de fabricação, consumo de combustível, etc) são chamadas de propriedades. Esse aspecto da POO permite a criação de diferentes objetos a partir de um mesmo modelo (classe), cada um com suas características próprias mas que, por mais diferentes que sejam, não deixam de ser objetos da mesma classe. Todos os objetos que pertencem a uma classe compartilham os mesmos atributos (MENDES, 2010).

Métodos

O terceiro ponto são os métodos desse objeto. Assim como as características indicam o que o objeto **é**, os métodos representam o que o objeto **faz**. São funções que, quando declaradas na classe, serão implementadas por todos os objetos dessa classe. No caso do carro, os métodos podem ser acendeFarol(), buzina(), entre outros. Os métodos (ou comportamento) são algo que o objeto realiza em resposta a um estímulo (LAFORE, 2001).

2.1.2 Encapsulamento

O segundo pilar é o encapsulamento, que adiciona uma camada extra de segurança ao sistema. O encapsulamento significa que as propriedades e métodos dos objetos são "escondidos" dentro dos mesmos, podendo ser acessados apenas através de métodos especiais (getters e setters, por exemplo). No exemplo do carro, o carro liga quando o motorista gira a chave, mesmo que quem tenha girado a chave não faça a menor ideia de como isso acontece internamente. O encapsulamento evita alterações indevidas, além de simplificar a elaboração, depuração e manutenção do programa (MENDES, 2010).

2.1.3 Herança

O pilar que permite a flexibilidade da Programação Orientada a Objetos é o conceito de herança, que permite que uma classe mais geral e abrangente seja especializada em subclasses que herdam todas as suas características, mas que podem implementar suas próprias propriedades independente da classe pai. Uma subclasse é considerada uma versão especializada ou extendida de sua classe pai e, consequentemente, de seus predecessores (RICHARD WIENER, 2000).

2.1.4 Polimorfismo

O quarto pilar da POO é o Polimorfismo. Às vezes, classes que descendem do mesmo pai necessitam realizar a mesma função de maneiras diferentes. Para isso, essa função pode ser declarada como abstrata na classe pai e implementada de acordo com as necessidades das classes filhas. A função ligar() de um carro elétrico é diferente da função ligar() de um carro convencional. Uma implementação possível para resolver esse problema é uma classe abstrata carro que possui uma função abstrata ligar(). As subclasses CarroEletrico e CarroConvencional são herdeiras da classe carro(), mas as respectivas funções ligar() são implementadas de maneiras diferentes em cada subclasse. Uma função sobrescrita aparenta realizar diferentes atividades dependendo do tipo de dado enviado a ela (LAFORE, 2001).

2.1.5 Composição

O quinto e último pilar é a Composição, que permite que objetos de uma classe façam parte de objetos de outra classe. A composição é uma forma mais forte de agregação. Até

mesmo um único objeto pode ser relacionado a uma classe através da composição. Em um carro, há apenas um motor, por exemplo (LAFORE, 2001).

2.1.6 Vantagens e Desvantagens

As principais vantagens da POO são a reutilização de código, facilidade de leitura e manutenção desse código e a facilidade de criação de bibliotecas, quando comparada a linguagens estruturadas. Por outro lado, como o desenvolvimento de algoritmos se torna mais complexo à medida em que novos membros são adicionados à equipe de desenvolvimento e à medida em que o projeto se torna maior, a POO geralmente diz respeito a estruturas também mais complexas (mas que são capazes de realizar tarefas mais complexas) do que estruturas de linguagens estruturadas (LAFORE, 2001).

2.2 Segurança da Informação

O mundo em que vivemos é digital. Isso é um fato. A transição do mundo analógico para o mundo digital foi uma revolução extremamente rápida e ocorreu em um curto período de tempo, mas foi-se a época em que falar que "o mundo tem se tornado cada vez mais digital" fazia sentido. O mundo é digital e aqueles que tentam resistir às mudanças estão fadados a ser superados pelos concorrentes. Até mesmo na vida pessoal é praticamente impossível se manter alheio às novas tecnologias.

Com essa revolução, surgiu também uma preocupação com a segurança dos dados e informações que, muitas vezes, fazem parte de sistemas acessados por milhões de pessoas diariamente e trafegam entre diferentes serviços e plataformas. Profissionais responsáveis por essa área devem estar constantemente se atualizando porque, apesar de os métodos de segurança serem atualizados e modernizados cada vez mais, as pessoas que estão atrás dos dados (e que estão determinadas a não deixar que nada as impeça de acessá-los) também se adequam. Conforme citado por Dangler (2013), alguns dos ataques mais famosos foram detectados após o dano ter sido feito, o que indica que as medidas de segurança são tomadas, muitas vezes, retroativamente. Isso pode indicar, também, que os métodos de ataque se modernizam mais rapidamente do que os mecanismos de defesa.

Falar sobre segurança vai muito além da senha você utiliza para acessar o e-mail ou o padrão utilizado para desbloquear seu celular. Por trás dessas ferramentas, existe uma série de mecanismos - uns mais complexos que outros - que visam a assegurar a segurança dos dados ali armazenados, e é papel do responsável pelo sistema a garantia dessa segurança.

2.2.1 Pilares

A segurança da informação tem sustentação em três pilares: confidencialidade, integridade e disponibilidade. Segundo Weidman (2014), um dos objetivos de um programa de

segurança da informação é definir o que é necessário para que se preserve determinado nível de confidencialidade, integridade e disponibilidade dos sistemas e dados relacionados à tecnologia da informação de uma empresa. Cada um desses pilares são importantes para a proteção de dados, e todas as políticas relacionadas à tecnologia da informação devem ser voltadas a garantir que os processos ocorram corretamente e em concordância com esses três pilares, que serão explicados a seguir.

2.2.1.1 Confidencialidade

A confidencialidade está relacionada à privacidade dos dados armazenados em um sistema. Assegurar a confidencialidade significa garantir que apenas entidades autorizadas tenham acesso a esses dados. Brechas na confidencialidade são extremamente danosas às organizações. Vazamentos de dados sigilosos sempre se traduzem em perdas financeiras e, muitas vezes, em problemas legais. A confidencialidade foca em previnir acesso não autorizado à informação armazenada (STAMP, 2005).

2.2.1.2 Disponibilidade

A disponibilidade é assegurada quando garante-se que os dados são acessíveis o tempo todo, ou seja, quando eles podem ser consultados, quando necessário, para uso autorizado (SCHUMACHER et al., 2005). Ataques de DoS (Denial of Service) são ataques que, ao deixar um sistema indisponível, afetam este pilar desse sistema. Assim como ocorre nos outros dois pilares, falhas de disponibilidade podem acarretar em prejuízos financeiros, multas, perda de credibilidade, entre outros danos.

2.2.1.3 Integridade

O terceiro pilar está intimamente relacionado ao primeiro. A integridade corresponde à garantia da confiabilidade e consistência dos dados armazenados, e é a proteção dos dados de uma empresa contra modificações não autorizadas (DANGLER, 2013). A falta de integridade significa que não se pode garantir a confidencialidade. Da mesma forma, sem confidencialidade não se pode garantir a integridade. Basicamente, uma informação possui integridade se a alteração de dados por entidades não autorizadas é proibida (STAMP, 2005).

2.3 Padrões de Projeto

O conceito de padrões de projeto foi introduzido em 1977 pelo arquiteto Cristopher Alexander, no âmbito arquitetônico, para designar soluções testadas e consolidadas que pudessem ser reutilizadas em vários projetos diferentes.

Cada padrão descreve um problema que ocorre repetidas vezes no nosso ambiente, para então descrever o núcleo da solução para aquele problema, de modo a possibilitar sua

utilização um milhão de vezes sem fazê-lo da mesma maneira uma vez sequer (CHRISTOPHER ALEXANDER, 1977).

A aplicação do termo para se referir a elementos de programação ocorreu, pela primeira vez, em 1987 por Beck et al (DOUGHERTY et al., 2009). Segundo Blakley e Heath (2004), Brad Appleton definiu um padrão como um pedaço de informação instrutiva que captura a estrutura essencial e a instrospecção de um grupo bem sucedido de soluções comprovadas de um problema recorrente que surge com um certo contexto e sistema de forças

Padrões de projeto, portanto, são soluções testadas e comprovadas para problemas recorrentes no processo de desenvolvimento de um software. No entanto, um padrão não é um um pedaço de código pronto para uso, mas sim um *template* de como resolver determinado problema, podendo ser adaptado à realidade de sua utilização. Segundo Dougherty et al. (2009), algoritmos não são considerados padrões de projeto, uma vez que solucionam problemas computacionais, e não problemas de *design*.

Padrões de projeto facilitam a reutilização de *designs* e arquiteturas bem sucedidos. Expressar técnicas comprovadas como padrões de projeto as tornam mais acessíveias a desenvolvedores de novos sistemas. Padrões de projeto auxiliam a escolha de alternativas de design que tornam o sistema reutilizável e evitam alternativas que comprometem a reusabilidade. Padrões de projeto podem, inclusive, melhorar a documentação e manutenção de sistemas já existentes, fornecendo uma especificação explícita das interações das classes e objetos e seus objetivos subjacentes. Resumindo, padrões de projeto auxiliam um designer a 'acertar' um design mais rapidamente (GAMMA et al., 1994).

2.3.1 Gang of Four

Em outubro de 1994, foi lançado o livro (muito provavelmente) mais importante sobre padrões de projeto até hoje. Escrito por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides (que ficaram conhecidos por Gang of Four ou GoF), o livro "Design Patterns: Elements of Reusable Object-Oriented Software" tinha seu objetivo dividido em duas partes: apresentar o conceito de padrões de projeto e apresentar um catálogo dos padrões de projeto levantados pelos autores.

Segundo Gamma et al. (1994), um padrão de projeto geralmente apresenta quatro elementos essenciais:

Nome

O nome é um identificador, utilizado para descrever um problema de design, suas soluções e consequências em uma ou duas palavras.

Problema

O problema descreve quando aplicar o padrão, explicando o problema e seu contexto.

Solução

A solução descreve os elementos que compõem o design, suas relações, responsabilidades e colaborações. Vale lembrar que a solução não descreve uma implementação específica, uma vez que o padrão nada mais é do que um template que pode ser utilizado em diversas situações

Consequências

As consequências são os resultados, vantagens e desvantagens da utilização do padrão.

Os autores separam os padrões apresentados em 3 grupos: criacionais, estruturais e comportamentais. Ao todo, são apresentados 23 padrões de projeto, sendo 5 criacionais, 7 estruturais e 11 comportamentais.

Os padrões de projeto criacionais focam no processo de criação dos objetos, e são representados pela Figura 1

Figura 1 – Padrões de Projeto Criacionais segundo GoF

Abstract Factory
Builder
Factory Method
Prototype
Singleton

Fonte: Gamma et al. (1994)

Por sua vez, os padrões estruturais lidam com a composição de classes ou objetos. Esses padrões são representados na Figura 2

Figura 2 – Padrões de Projeto Estruturais segundo GoF

Adapter
Bridge
Composite
Decorator
Façade
Flyweight
Proxy

Fonte: Gamma et al. (1994)

Por fim, os padrões comportamentais são aqueles que caracterizam as maneiras como as classes ou objetos interagem e distribuem responsabilidades. A Figura 3 representa esses padrões.

Figura 3 – Padrões de Projeto Comportamentais segundo GoF

Chain of Responsibility
Command
Interpreter
Iterator
Mediator
Memento
Observer
State
Strategy
Template Method
Visitor

Fonte: Gamma et al. (1994)

Além de apresentar os padrões, o GoF também introduz um template para a descrição dos padrões (explicado na seção de Metodologia e utilizado neste trabalho). O livro é, até hoje, considerado como um *must-read* de todo engenheiro de software, mantendo-se relevante mesmo após quase 22 anos.

2.3.2 Padrões de Segurança

Existem inúmeros padrões de projeto que se propõem a resolver diversos tipos de problema. Um desses tipos são os problemas relacionados à segurança da informação. No entanto, a literatura que aborda esse tipo específico de solução não é tão abundante quanto se espera, tendo em vista a importância que essa área tem.

Esse cenário é um tanto quanto contraditório, dado a importância do papel da segurança em um sistema de informação. Soluções já testadas e validadas são aliados essenciais contra ataques, que ocorrem a todo momento no mundo inteiro. Além disso, como Dougherty et al. (2009) afirma, o custo de consertar vulnerabilidades de um sistema e os riscos associados a vulnerabilidades após a implementação do sistema são grandes tanto para desenvolvedores quanto para usuários finais. Apesar disso, Halkidis, Chatzigeorgiou e Stephanides (2004) afirmam que apenas recentemente foi reconhecido que a principal fonte dos ataques que questionam as características de segurança de sistemas de informação é, na maioria dos casos, softwares mal projetados e desenvolvidos.

Geralmente, a segurança de um sistema é ignorada inicialmente porque ou a política de segurança não é disponível, ou apenas por parecer mais fácil adiar preocupações com a segurança. Ignorar questões de segurança é perigoso porque pode ser difícil ajustar a segurança de uma aplicação retroativamente. Mesmo que o design de uma aplicação, inicialmente, possa ser mais complicado por incorporar a segurança desde o princípio, o design será mais limpo do que o resultado da integração da segurança já durante o ciclo de desenvolvimento. (YODER; BARCALOW, 1997)

Padrões de projeto de segurança, de acordo com Elder (2001), são soluções bem compreendidas para um problema recorrente relacionado à segurança da informação e, se utilizados, ajudam a diminuir os riscos relacionados à segurança de um sistema.

Os artigos existentes sobre o tema podem ser separados basicamente em dois grupos: os artigos que se preocupam em organizar/classificar, de alguma forma, os padrões, e os artigos que se propõem a apresentar alguns padrões, muitos deles utilizando o template proposto por Gamma et al. (1994).

Em seu trabalho, considerado o pioneiro na área de padrões de projeto de segurança, Yoder e Barcalow (1997) apresentam 7 padrões que podem ser implementados ao desenvolver as questões de segurança de uma aplicação. Em um dos mais completos trabalhos, Schumacher et al. (2005) apresentam, de forma detalhada, 46 padrões de projeto relacionados à segurança. Dangler (2013), por sua vez, apresenta 30 padrões e implementa alguns em um estudo de caso utilizando um sistema da Universidade do East Tennessee. Halkidis, Chatzigeorgiou e Stephanides (2004)

apresenta diagramas de classe e uma breve descrição do funcionamento de 13 padrões. Após apresentar 9 padrões, Rosado, Fernández-Medina e Piattini (2006) propõem um framework para compará-los. Romanosky (2001), Kienzle et al. (2001), Blakley e Heath (2004), Dougherty et al. (2009) e Röser (2012) também têm como objetivo a apresentação dos padrões utilizando templates semelhantes ao proposto por Gamma et al. (1994). Netland, Espelid e Mughal (2006), por sua vez, apresentam, de maneira minuciosa, apenas um padrão: o input validator.

Quando combinados, a literatura utilizada para a realização deste trabalho aponta a existência de cerca de 250 padrões. Porém, muitos padrões têm um funcionamento muito semelhante e, muitas vezes, redundante. A falta de padronização desses padrões acaba contribuindo para que um mesmo padrão seja descrito mais de uma vez com mais de um nome. Não há um repositório oficial para esses padrões e, segundo uma análise feita por P. Ponde, S. Shirwaikar e Gore (2016) ao realizarm uma comparação de 52 padrões, constataram que 34 (65%) desses eram redundantes. Essa análise constatou até 9 padrões idênticos que tinham os mesmos atributos e, portanto, foram considerados como reduntantes. Além disso, quase metade dos padrões levantados não possuem sequer seu objetivo identificado, 40% dos padrões possuem uma explicação minimamente suficiente para a compreensão do seu escopo, e apenas 66 (pouco mais de 25%) apresentam diagramas UML que elucidem seu funcionamento. Através de seu trabalho, Heyman et al. (2007) concluiram que a falta de soluções bem descritas indica que uma parte significativa dos padrões analizados por ele são mais focados no problema do que na solução em si.

Essa proporção de número de padrões/quantidade de informação se dá porque os artigos que citam a existência do maior número de padrões apenas o fazem para tentar separá-los por categorias, mas não se preocupam em, de fato, apresentá-los. Heyman et al. (2007) fazem uma análise do panorama dos padrões de projeto de segurança, sem se preocupar muito em citá-los. Por outro lado, P. Ponde, S. Shirwaikar e Kreiner (2016) e P. Ponde, S. Shirwaikar e Gore (2016), artigos com um autor em comum, analisam os padrões utilizando *clusters* (cujas características foram determinadas pelos autores). 210 padrões são citados por esses artigos, mas não são apresentadas mais informações além do nome da maioria. Bunke, Koschke e Sohr (2012) também apenas cita um grande número de padrões (163), mas sem apresentar mais informações. Elder (2001), por sua vez, cita, sem fornecer muitos detalhes, 29 padrões que são apresentados, de fato, por Kienzle et al. (2001).

Alguns artigos, no entanto, fogem dessa dicotomia. Rimba et al. (2015), por exemplo, propõe uma abordagem nova para a composição de padrões cujo objetivo é a construção de sistemas seguros. Muijnck-Hughes e Duncan (2013) identificam uma falta de padronização que afeta tanto o desenvolvimento de novos padrões quanto a classificação dos padrões existentes. Jafari e Rasoolzadegan (2018) fazem uma análise do estado da arte dos padrões de projeto, focando nos trabalhos existentes e sua qualidade. Hafiz, Adamczyk e Johnson (2012), por sua vez, apresentam uma espécie de dicionário para os padrões de segurança existentes. Por fim, Mahmoud (2000) apresenta um único padrão que tem como objetivo garantir a segurança de um

dispositivo que executa um mobile code.

2.3.2.1 Categorização Atual

Um consenso nos artigos analisados é, ironicamente, a falta de consenso em relação à categorização que deve ser adotada para abranger satisfatoriamente os padrões de segurança existentes. Segundo Muijnck-Hughes e Duncan (2013), a grande variedade de templates utilizados na criação dos padrões dificulta a classificação, identificação e comparação dos padrões semelhantes. P. Ponde, S. Shirwaikar e Kreiner (2016) defendem, em um dos diversos trabalhos que propõem uma nova métrica de classificação, que isso se dá porque tentativas de organizar os padrões resultaram em várias metodologias para classificação. Em sua publicação, Heyman et al. (2007) vão além e afirmam que alguns padrões não devem sequer ser considerados como padrões por serem abstratos demais e, por fim, sugerem a adoção de uma documentação padrão para os padrões de projeto de segurança. P. S. Ponde, S. C. Shirwaikar e Kharat (2017) defendem que a grande variedade de categorias utilizadas para a classificação não é benéfica e dificulta a escolha do padrão adequado para resolver um determinado problema de segurança.

É possível notar, portanto, que um problema recorrente relacionado aos padrões de projeto de segurança é a ausência, até então, de um método classificatório oficial para que os padrões possam ser identificados e organizados de uma maneira consistente. No entanto, alguns métodos de classificação são mais recorrentes na maioria dos artigos que fazem um levantamento de como os padrões têm sido classificados na literatura.

Quinze artigos categorizam, de algum jeito, os padrões. Ao analisar os critérios utilizados para fazer a classificação, dois pontos chamam a atenção e podem ajudar a explicar a falta de um padrão a ser utilizado para essa classificação. O primeiro é a quantidade de métodos de classificação existentes. Ao todo, são 22 métodos, citados nesses 15 artigos. Desses 22 métodos, mais da metade é citada por apenas um artigo (P. S. Ponde, S. C. Shirwaikar e Kharat (2017), por si só, cita 13 métodos). O segundo ponto é a falta de padronização até para identificar os diferentes métodos utilizados.

Algumas classificações se referem aos mesmos parâmetros, mas com nomes diferentes. Outras têm os mesmos nomes mas diferem um pouco nos parâmetros. O método de classificação que utiliza o nível da aplicação (se é core, perimeter ou exterior), por exemplo, é referido como classificação por Location P. Ponde, S. Shirwaikar e Kreiner (2016) e P. S. Ponde, S. C. Shirwaikar e Kharat (2017), Application Context Munawar Hafiz (2006) e Hafiz, Adamczyk e Johnson (2007), ou ainda apenas pelos seus parâmetros (core x perimeter x exterior). O mesmo acontece para vários outros métodos de classificação. Já o método que classifica de acordo com os Objetivos/Atributos de Segurança nos quais o padrão propõe focar tem como parâmetros *Access Control, Software Integrity, Autenthication, Confidentiality, Authorization, Identification, Data Integrity, Security Auditing, Attack/Harm Detection, Privacy, Non-repudiation, Anonymity, Availability, Recovery e Prosecution* em Heyman et al. (2007), *Authentication/Identity, Access Control/Authorization, Integrity, Confidentiality, Availability, Non-Repudiation, Accountability*

e Key Management em P. Ponde, S. Shirwaikar e Kreiner (2016) mas apenas *Integrity, Confidentiality e Availability* em Hafiz, Adamczyk e Johnson (2007) (onde é chamado de modelo da CIA).

Essa falta de padronização prejudica não só desenvolvedores que buscam qual padrão utilizar, mas também dificulta a realização de pesquisas que poderiam ajudar a criar um método conciso de classificação de padrões de projeto de segurança. A lista com todos os métodos de classificação encontrados é a seguinte:

Baseados em princípios de segurança

Citado por: Munawar Hafiz (2006), Heyman et al. (2007), Hafiz, Adamczyk e Johnson (2007), Dangler (2013), P. Ponde, S. Shirwaikar e Kreiner (2016), P. Ponde, S. Shirwaikar e Gore (2016) e P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

Heyman et al. (2007) se refere a essa categoria como Objetivos de Segurança e suas subcategorias são: *Access Control, Software Integrity, Autenthication, Confidentiality, Authorization, Identification, Data Integrity, Security Auditing, Attack/Harm Detection, Privacy, Non-repudiation, Anonymity, Availability, Recovery e Prosecution*

Já P. Ponde, S. Shirwaikar e Kreiner (2016) se refere a ela como Atributos da Qualidade de Segurança com as seguintes subcategorias: *Authentication/Identity, Access Control/Authorization, Integrity, Confidentiality, Availability, Non-Repudiation, Accountability* e *Key Management*.

Dangler (2013) se refere como requisitos não funcionais de segurança para a implementação de um sistema e suas subcategorias são: Accountability, Availability, Confidentiality e Integrity.

P. Ponde, S. Shirwaikar e Gore (2016), por sua vez, se refere como Princípios de Segurança com as seguintes subcategorias (as mesmas de P. Ponde, S. Shirwaikar e Kreiner (2016)): *Authentication, Access Control, Integrity, Confidentiality, Availability, Non-Repudiation, Accountability* e *Key Management.*

P. S. Ponde, S. C. Shirwaikar e Kharat (2017) também se refere a ela como Princípios de segurança, e cita *Confidentiality, Integrity, Authentication, Authorization, Availability, Confidentiality, Access Control, Non-repudiation* e *Accountability*.

Hafiz, Adamczyk e Johnson (2007) se refere a esse modelo como Modelo da CIA, com apenas 3 subcategorias: *Confidentiality, Integrity e Availability*.

Por fim, Munawar Hafiz (2006) se refere a esse modelo como classificação baseada em conceitos de segurança e suas subcategorias são as mesmas de Dangler (2013): *Accountability, Availability, Confidentiality* e *Integrity*.

Baseados em Propósito

Citado por: Elder (2001), Kienzle et al. (2001), Munawar Hafiz (2006), P. Ponde, S. Shirwaikar e Kreiner (2016), P. Ponde, S. Shirwaikar e Gore (2016) e P. S. Ponde, S. C.

Shirwaikar e Kharat (2017).

P. Ponde, S. Shirwaikar e Kreiner (2016) e P. Ponde, S. Shirwaikar e Gore (2016) se referem a esse método de maneiras diferentes (propósito e tipo, respectivamente), mas com as mesmas subcategorias: *Structural, Behavioral* e *Generic*.

P. S. Ponde, S. C. Shirwaikar e Kharat (2017) se refere como Purpose e considera 5 subcategorias: *Structural, Behavioural, Generic, Procedural* e *Creational*.

Elder (2001) e Kienzle et al. (2001) não dão nome ao método, mas separam os padrões apresentados em *Structural* e *Procedural*. Munawar Hafiz (2006) também utiliza as mesmas subcategorias, mas se refere a essa classificação como 'baseada em produto e processo'.

Baseados na Localização

Citado por: Munawar Hafiz (2006), Hafiz, Adamczyk e Johnson (2007), P. Ponde, S. Shirwaikar e Kreiner (2016), P. Ponde, S. Shirwaikar e Gore (2016) e P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

Os 5 artigos utilizam as mesmas subcategorias, mas dão nomes diferentes à classificação: *Application Context, Application Context, Location, Trust Boundary* e *Location*, respectivamente.

Baseados no Modelo de Ameaça (STRIDE)

Citado por: Halkidis, Chatzigeorgiou e Stephanides (2004), Munawar Hafiz (2006), Hafiz, Adamczyk e Johnson (2007), P. Ponde, S. Shirwaikar e Kreiner (2016), P. Ponde, S. Shirwaikar e Gore (2016) e P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

STRIDE é a sigla utilizada para representar 6 tipos de ameaça existentes no âmbito da segurança da informação: *Spoofing, Tampering, Repudiation, Information disclosure, Denial of service* e *Elevation/Escalation of privilege*. Todos os artigos que citam essa classificação concordam em relação às subcategorias.

Baseados no Tipo do Sistema

Citado por: Halkidis, Chatzigeorgiou e Stephanides (2004), Blakley e Heath (2004), Munawar Hafiz (2006) e P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

Classificação proposta por Blakley e Heath (2004), com 2 subgrupos: *Available System Patterns* (que auxiliam a construção de sistemas que asseguram a disponibilidade do sistema) e *Protected System Patterns* (que auxiliam a construção de sistemas que protegem os recursos contra acessos não autorizados, ou seja, assegura a confidencialidade e, consequentemente, a integridade do sistema).

Baseados nos Estágios do Ciclo de Vida

Citado por: Dougherty et al. (2009), Dangler (2013), P. Ponde, S. Shirwaikar e Kreiner (2016), P. Ponde, S. Shirwaikar e Gore (2016) e P. S. Ponde, S. C. Shirwaikar e Kharat (2017)

Dougherty et al. (2009) e Dangler (2013) se referem a esse modelo como Nível de Design, mas citam os mesmos subgrupos: Nível Arquitetural, de Design e de Implementação.

P. Ponde, S. Shirwaikar e Kreiner (2016) e P. Ponde, S. Shirwaikar e Gore (2016) se referem a esse modelo como Estágios do Ciclo de Vida e citam, além dos subgrupos anteriores, o estágio de requisitos.

P. S. Ponde, S. C. Shirwaikar e Kharat (2017) também se refere a esse modelo como Estágios do Ciclo de Vida, mas cita 10 subgrupos: Arquitetural, Requisitos, Análise, Design, Implementação, Integração, Deployment, Operação, Manutenção e Disposal.

Baseado em Tiers

Citado por: Steel, Nagappan e Lai (2003), P. S. Ponde, S. C. Shirwaikar e Kharat (2017) e Munawar Hafiz (2006).

Steel, Nagappan e Lai (2003) separa os padrões em 4 tiers: Web, Business, Web Service e Identity tiers.

P. S. Ponde, S. C. Shirwaikar e Kharat (2017) chama essa classificação de 'Logical Tiers' e separa em: *Application, Host, Client, Logic, Data, Integration, Platform and OS, Distribution, Transport* e *Network*.

Munawar Hafiz (2006) também se refere a essa classificação como Logical Tiers, mas separa em Presentation/WEB, Business e Integration tier.

Baseado em Enterprise Level Security

Citado por: Röser (2012) e P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

P. S. Ponde, S. C. Shirwaikar e Kharat (2017) divide em 9 subcategorias: *Enterprise Security and Risk Management, Identification & Authentication, Access Control, System Access Control, Operating System Access Control, Accounting Patterns, Firewall Architecture, Secure Internet Applications* e *Cryptographic Key Management*.

Röser (2012), apesar de não citar o nome do método de classificação, divide em subgrupos semelhantes aos utilizados por P. S. Ponde, S. C. Shirwaikar e Kharat (2017): *ID Management, Access Control, Accounting, Operating System, Client-Server* e *Securing Applications*.

Baseado no modelo da Microsoft

Citado por: Munawar Hafiz (2006) e Hafiz, Adamczyk e Johnson (2007).

Segundo Munawar Hafiz (2006), o esquema de classificação de padrões de projeto proposto pela Microsoft consiste em uma tabela bidimensional que encapsula os diferentes

aspectos da arquitetura de software. As linhas representam os stakeholders (Arquiteto, Designer e Developer), ao passo que as colunas representam os aspectos em si: função, dados e testes.

Baseado em Interrogativas

Citado por: P. S. Ponde, S. C. Shirwaikar e Kharat (2017)

P. S. Ponde, S. C. Shirwaikar e Kharat (2017) se refere a essa classificação como classificação por interrogativas, cujos parâmetros são: *Purpose (Why), Data (What), Function (How), Timing (When), Network (Where), People (Who)* e *Scorecard (Test)*.

Baseado no framework de Zachman

Citado por: Munawar Hafiz (2006).

Classificação relacionada à classificação baseada em interrogativas citada por P. S. Ponde, S. C. Shirwaikar e Kharat (2017). O Framework consiste em uma tabela bidimensional em que as colunas são as interrogativas, ao passo que as linhas são os elementos que compõem a concepção de um sistema (Escopo, Modelo de Negócio, Modelo de Sistema, Modelo de Tecnologia, Representação Detalhada e Sistema Funcional). Esse framework serviu de base para o modelo da Microsoft, discutido anteriormente.

Baseado em STRIDE X Localização

Citado por: Munawar Hafiz (2006) e Hafiz, Adamczyk e Johnson (2007).

Classificação representada por uma árvore que aplica os elementos do STRIDE nos três níveis/localização do sistema (*Core, Perimeter* e *Exterior*)

Baseado nos 10 princípios guias de McGraw

Citado por: Halkidis, Chatzigeorgiou e Stephanides (2004).

Os princípios são:

1. Proteger o elo mais fraco, uma vez que é o componente do sistema mais suscetível a um ataque bem sucedido;

2. Praticar defesa em profundidade, ou seja, aplicar vários níveis de defesa de modo a aumentar a chance de detecção de erros;

3. O sistema deve falhar de maneira segura, o que significa que o sistema deve continuar a operar em modo de segurança em caso de falha;

4. Seguir os princípios do menor Privilégio: deve-se conceder, pelo menor tempo possível, o menor nível de acesso possível para que uma ação seja realizada;

5. Compartimentalizar, minimizando o dano que pode ser causada a um sistema através da separação do mesmo no menor número de unidades possível ao mesmo tempo que isola o código que contém os privilégios de segurança;

6. Manter o sistema limpo. Sistemas complexos são mais propensos a possuir falhas de segurança;

7. Promover a privacidade;

8. Construir sistemas em que até mesmo ataques internos sejam difíceis;

9. Não confiar em softwares que não tenham sido extensivamente testados;

10. Usar soluções consolidadas e bem testadas;

Segundo Halkidis, Chatzigeorgiou e Stephanides (2004), McGraw descreve esses 10 princípios como um guia para a construção de softwares seguros.

Baseado em Brechas

Citado por: Halkidis, Chatzigeorgiou e Stephanides (2004).

Divisão de acordo com critérios que descrevem quão bem um padrão impede os desenvolvedores de construirem um sistema que contenha brechas de segurança comuns. O artigo foca em 3 brechas: *Buffer Overflow, Poor Access Control Mechanisms* e *Race Condition*

Baseado em Tipos de Padrão

Citado por: Heyman et al. (2007).

Separa os padrões em 3 categorias: *Core Patterns, Guideline Principles* e *Process Activities*

Baseado em Resposta

Citado por: P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

Consiste em 7 subcategorias: *Avoidance, Deterrence, Prevention, Detection, Mitigation, Recovery* e *Forensics*.

Baseados em Tipos de Código Fonte

Citado por: P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

Consiste em 10 subcategorias: *New, Open-source Runtime, Model transform, Wizard code, Reuse library, Outsourced, Legacy, Off-the-shelf* e *Remote web service*.

Baseado em Domínio

Citado por: P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

Separa em: Web Services, J2EE e Microsoft.

Baseado em Restrições

Citado por: P. S. Ponde, S. C. Shirwaikar e Kharat (2017).

Divide em: *Regulatory, Organizational, Human* e *Mechanism*

Baseado na Classificação Hierárquica

Citado por: Bunke, Koschke e Sohr (2012).

Classificação representada por uma tabela bidimensional cujas linhas representam os elementos dos Princípios de Segurança e as colunas representam os três propósitos dos padrões: Estrutural, Comportamental e Conceito Genérico.

Baseado na Roda de Segurança

Citado por: Munawar Hafiz (2006).

A classificação utilizando a roda apresenta os atributos de segurança representados pelos aros da roda e, no centro, está o serviço ou aplicação que está sob consideração. Os limites internos da roda representam os perímetros de segurança.

Baseado no Cubo de McCumber

Citado por: Munawar Hafiz (2006).

O cubo de McCumber é uma representação tridimensional que combina 3 tipos de elementos e suas 3 categorias: categorias primárias de proteção (tecnologia, práticas/políticas e fator humano), estados da informação (transmissão, armazenamento e processamento) e pilares da segurança da informação (integridade, confidencialidade e disponibilidade).

Baseado em DREAD

Citado por: Munawar Hafiz (2006).

DREAD é um mecanismo de cálculo de risco. Cada letra representa um atributo de possíveis ameaças.

Damage Potential: o dano que será causado caso a vulnerabilidade seja explorada por um atacante.

Reproductibility: a facilidade de explorar essa vulnerabilidade.

Exploitability: a habilidade necessária para explorar a vulnerabilidade.

Affected Users: as partes afetadas por um ataque

Discoverability: facilidade de exploração e descoberta de uma vulnerabilidade do sistema.

Os métodos mais citados foram os métodos baseados nos princípios de segurança, propósitos e no modelo STRIDE (7, 6, e 6 citações respectivamente). Apesar da grande quantidade de métodos levantados, vale notar que apenas uma pequena parte deles foi usada, de fato, para classificar padrões. Métodos menos difundidos (principalmente aqueles com menos citações) não apresentaram, ao menos na literatura analisada, exemplos práticos de classificação de padrões de projeto de segurança.

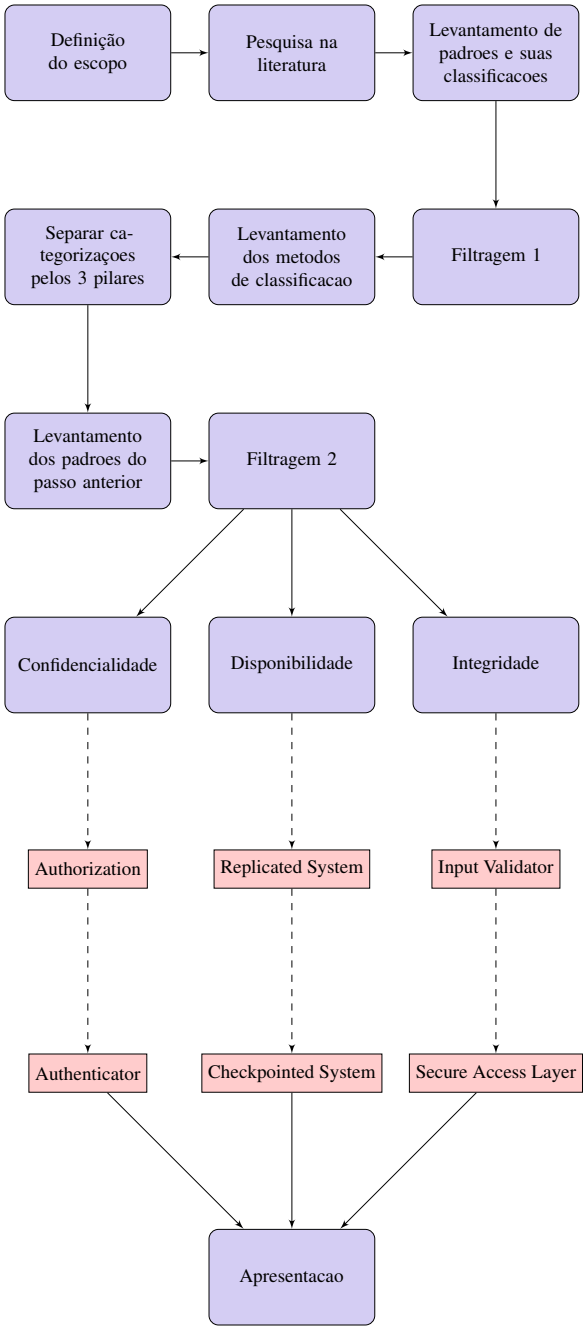
Apenas através da análise dos métodos citados, é possívEl perceber que, além de não existir um método específico para a classificação dos padrões, a grande quantidade de métodos existentes e a falta de um padrão para classificar os próprios métodos dificulta a adoção de um método definitivo. Com isso em mente, para dar continuidade ao trabalho, a classificação considerada como a mais relevante foi a baseado nos princípios da segurança da informação, mais especificamente nos 3 pilares: disponibilidade, integridade e confidencialidade.

A escolha desse método tem dois motivos: primeiro, ele foi o método mais citado pelos artigos que mencionavam métodos de classificação de padrões de projeto de segurança; e segundo, os 3 pilares representam um aspecto mais palpável da segurança da informação, principalmente por serem elementos ensinados em aulas de segurança e auditoria de sistemas. A utilização dos três pilares une conceitos de programação orientada a objetos, padrões de projeto e segurança da informação de uma forma que pode fornecer, a quem esteja pesquisando a respeito do assunto, um exemplo prático aplicável ao mundo real.

3 MATERIAL E MÉTODOS

A metodologia utilizada para a realização deste trabalho foi extensa e pode ser dividida em várias partes. A Figura 4 pode ser utilizada como referência para representar o fluxo seguido.

Figura 4 – Fluxograma da metodologia proposta para este trabalho.



Fonte: Gerada pelo autor

A primeira etapa para a realização deste trabalho foi a pesquisa por artigos e publicações em geral que dissessem respeito a padrões de projeto cujo foco fosse segurança da informação. Foram selecionados 26 artigos sobre o tema. Um ponto que vale mencionar (e que,

de certo modo, ajudou a moldar e direcionar o objetivo geral do trabalho) é a dificuldade em encontrar artigos que abordassem padrões de projetos voltados à segurança da informação em português. Sendo assim, todos os 26 artigos encontrados estavam escritos em inglês. Com isso em mente, a intenção foi construir um artigo em português que servirá como referência para a utilização de padrões de projeto de segurança.

Após uma primeira análise, foram levantados 248 padrões. Devido à falta de padronização na classificação dos padrões, cada artigo os classificava de uma forma diferente, escolhida peLos autores. Para melhor visualização, os padrões identificados foram organizados em uma planilha. As linhas da planilha representam cada padrão, ao passo que as colunas representam as características desse padrão, de acordo com cada artigo abordado. As 5 primeiras colunas representam características gerais dos padrões: objetivo, citações, outros nomes pelos quais o padrão pode ser conhecido, observações gerais e diagramas. As outras 22 colunas se referiam a categorizações específicas de acordo com cada artigo abordado. Uma Tabela com a representação simplificada dessa planilha encontra-se no apêndice (Tabela 3).

Essa planilha possibilitou uma visão ampla dos padrões existentes e suas características. No entanto, essa visão se mostrou superficial, uma vez que a planilha apresentava apenas aspectos gerais dos padrões, não abrangendo detalhes de uso, implementação, vantagens e desvantagens, entre outros fatores que são de suma importância para quem deseja utilizá-los ou apenas estudá-los melhor. Em um primeiro momento, o objetivo inicial do trabalho seria propor um método concreto para a classificação de todos os padrões encontrados. Esse objetivo se mostrou inviável, uma vez que, após análise da planilha, constatou-se que a maioria dos padrões era apenas citada, então muitas vezes tudo o que se sabia sobre aquele padrão era seu nome. Outro fator que desencorajou a proposta desse método foi a grande quantidade de artigos que já haviam seguido esse caminho. Nesse momento, o objetivo foi alterado, passando a focar na apresentação dos padrões mais relevantes dentre os levantados.

O próximo passo, portanto, consistiu em filtrar os padrões e classificá-los de acordo com sua relevância. Inicialmente, foram separados 15 padrões, classificados apenas pelo número de artigos que os citava. No entanto, há artigos que citam mais de 100 padrões, mas sem entrar em detalhes a respeito dos mesmos, ao passo que existem artigos que citam poucos, mas apresentam muito mais informações importantes sobre eles. Para evitar que a simples citação de um padrão por um artigo tivesse o mesmo valor que a explicação detalhada desse padrão por outro, foi feita uma reclassificação desses 15 padrões. Para isso, os artigos foram separados em dois grupos: os artigos quantitativos, que apenas citavam os padrões, e os artigos qualitativos, que os apresentavam de forma minimamente aprofundada e estruturada. A cada grupo de artigos foi assimilado um peso (.25 para quantitativos e 2 para qualitativos), que seria atribuído às citações de padrões de projeto por esses artigos. Outro ponto que pesou nessa cLassificação foi a descrição gráfica desses padrões: quanto mais informações em forma de diagramas de classe, sequência, ou imagens da estrutura do padrão, melhor, e cada diagrama tinha o peso 1. A Tabela 1 apresenta o resultado dessa primeira filtragem, com a classificação final dos 15 padrões

separados, considerados os mais relevantes dentre a lista de 248. A Tabela completa com as métricas utilizadas encontra-se no apêndice (Tabela 4).

Tabela 1 – Resultado da primeira filtragem

Padrão	Pontuação Final
Check Point	14
Input Validation	13.75
Single Access Point	13.25
Authenticator	12.25
Secure Visitor	11
Secure Access Layer	10.75
Secure Logger	10.25
Session/Secure Session	9.75
Authorization	9.25
Multilevel Security	9
Checkpointed System	8.25
Replicated System	8.25
Subject Descriptor	8.25
Roles	8
Minefield	3.5

Fonte: Gerada pelo autor

Nesse momento, foi decidido o objetivo final do trabalho: apresentar os padrões mais relevantes considerando cada um dos 3 pilares da segurança da informação: confidencialidade, integridade e disponibilidade. 6 padrões seriam apresentados (2 de cada pilar). Para isso, foi necessário uma nova divisão dos artigos em duas categorias: os artigos que categorizavam os padrões, e os artigos que não os categorizavam. Dentre os artigos que categorizavam, foi necessária uma segunda divisão: entre os artigos que incluíam a categorização de acordo com os 3 pilares e os artigos que não utilizavam essa categoria. Por fim, uma terceira divisão foi feita: alguns artigos apenas citavam a existência das categorias relacionadas aos 3 pilares, ao passo que outros artigos citavam quais padrões se encaixavam em qual pilar, o que era o foco do trabalho.

Dos 26 artigos iniciais, quatro artigos utilizavam a categorização em três pilares e citavam os padrões que encaixavam em cada uma. Uma análise foi feita desses artigos e o resultado dessa análise foi uma nova tabela: padrões de projeto citados pelos artigos que os categorizam de acordo com os 3 pilares da segurança da informação (Tabela 5).

Ainda restava selecionar os dois padrões mais relevantes de cada categoria. Para isso, a Tabela 1 (contendo os 15 padrões mais relevantes dentre todos os 248 padrões levantados inicialmente) foi comparada com a Tabela 2, gerada a partir dos artigos que categorizavam e citavam os padrões de acordo com os 3 pilares. O resultado foi, de certo modo, esperado: dos 15 artigos selecionados na Tabela 1, 13 também figuravam entre os 15 mais mencionados pelos artigos na Tabela 5. As únicas diferenças entre as duas filtragens consistiam em:

Padrões Secure Preforking e Clear Sensitive Information

Estava entre os mais citados na Tabela 5, mas não na Tabela 1, ou seja, eram padrões relativamente relevantes em relação aos padrões separados pela categoria dos 3 pilares, mas não eram relevantes no cenário geral dos padrões de projeto.

Padrão Secure Logger

Estava presente entre os mais citados na Tabela 1, mas não na Tabela 2. Ou seja, por mais que fosse um padrão teoricamente relevante entre todos os padrões de projeto de segurança, ele não havia sido citado por nenhum artigo que utilizava a métrica dos 3 pilares para categorizar os padrões.

Padrão Roles

Estava presente em ambas as filtragens, mas não estava entre os 15 mais citados da Tabela 5. Apesar de ter sido bastante citado pelos artigos de modo geral, apenas um artigo dos que utilizavam os pilares para classificar padrões citava esse padrão.

Por fim, os padrões que não estavam presentes em ambas as filtragens foram retirados, o padrão roles foi adicionado e o resultado da segunda filtragem foi uma lista de 14 padrões (quase idêntica à primeira filtragem, com exceção do Secure Logger). Os 14 padrões selecIonados e seus respectivos pilares são representados na Tabela 2. A Tabela completa com todos os padrões de cada um dos três pilares encontra-se no apêndice (Tabela 5)

Tabela 2 – Resultado da segunda filtragem

Padrões	Pilares		
	Disponibilidade	Confidencialidade	Integridade
Authenticator		x	
Authorization		x	
Checkpointed System	x		x
Minefield	x	x	x
Single Access Point		x	
Subject Descriptor		x	
Check Point		x	x
Input Validation			x
Multilevel Security		x	x
Replicated System	x		x
Secure Access layer			x
Secure Visitor		x	x
Session/Secure Session		x	x
Roles		x	

Fonte: Gerada pelo autor

É difícil separar e definir qual aspecto da segurança da informação é o foco de um padrão de projeto, já que os conceitos se sobrepõem e um pilar está, muitas vezes, relacionado aos outros dois. Portanto, como muitos padrões não se encaixam em apenas um pilar, ainda restava escolher 2 padrões por pilar para serem apresentados. Os padrões Input Validation e Secure Access Layer foram escolhidos como padrões de Integridade por serem os únicos com apenas essa categoria. Três foram classificados como padrões de Disponibilidade: Checkpointed System, Replicated System e Minefield. Por dois motivos, os escolhidos para esse pilar foram o Checkpointed System e o Replicated System: primeiro, o padrão Minefield foi classificado como um padrão pertencente aos 3 pilares; segundo: o padrão Minefield era muito citado, mas apenas 1 artigo (dos 7 que o citava) foi considerado um artigo qualitativo. Por fim, os padrões escolhidos para o pilar Confidencialidade foram o Authenticator e o Authorization.

Para a apresentação aprofundada dos padrões, foi utilizado o template proposto por Gamma et al. (1994), estruturado da seguinte maneira:

Nome e classificação do padrão

Nome do padrão e suas classificações de acordo com os 26 artigos analisados

Objetivo

Uma breve declaração que responde às seguintes perguntas: O que o padrão faz? Qual é o seu objetivo principal? Qual problema específico ele pretende resolver?

Also Known As (AKA)

Outros nomes pelos quais o padrão pode ser reconhecido, caso haja algum

Motivação

Um cenário que ilustre um problema de design e como as classes/objetos do padrão irão resolvê-lo. Esse cenário ajuda a entender melhor a descrição abstrata do padrão

Aplicabilidade

Quais são as situações em que o padrão de projeto pode ser utilizado? Quais são os exemplos de um design fraco que o padrão pode resolver? Como é possível reconhecer essas situações?

Estrutura

Uma representação gráfica das classes do padrão utilizando diagramas UML

Participantes

As classes e/ou objetos que participam do padrão e suas responsabilidades

Colaborações

Como ocorre a colaboração dos participantes para realizarem suas responsabilidades

Consequências

Quais são as consequências da utilização do padrão? Como ele garante seus objetivos? Quais são as trocas e resultados do uso do padrão? Qual aspecto da estrutura do sistema o padrão permite que você varie independentemente?

Implementação

A quais armadilhas, dicas ou técnicas você deve ficar atento na hora de implementar? Existe algum problema específico de alguma linguagem?

Exemplo

Exemplo de código

Usos conhecidos

Exemplos do uso do padrão encontrados em sistemas reais.

Padrões relacionados

Quais outros padrões se relacioNam com esse? Quais são as principais diferenças? Com qual outro padrão esse padrão deve ser utilizado?

Os padrões foram implementados, a título de exemplificação, utilizando a linguagem PHP e os resultados são apresentados no capítulo a seguir.

4 RESULTADOS E DISCUSSÃO

Dois padrões foram escolhidos para representar cada um dos três pilares da segurança da informação. Para Disponibilidade, foram selecionados os padrões Checkpointed System e Replicated System. Para Confidencialidade, Authorization e Authenticator. Por fim, para Integridade, os padrões escolhidos foram o Secure Access Layer e o Input Validator.

4.1 Disponibilidade

São apresentados, a seguir, os padrões considerados mais relevantes do pilar Disponibilidade: Checkpointed System e Replicated System.

4.1.1 Checkpointed System

Tem como objetivo armazenar imagens de estados válidos dos componentes de modo a permitir a recuperação em caso de falha.

Nome e classificação do padrão

Checkpointed System

Available System Halkidis, Chatzigeorgiou e Stephanides (2004) e Blakley e Heath (2004), Structural Bunke, Koschke e Sohr (2012), Integrity Bunke, Koschke e Sohr (2012), Availability Hafiz, Adamczyk e Johnson (2007) e Bunke, Koschke e Sohr (2012), Core Security Hafiz, Adamczyk e Johnson (2007), Tampering (STRIDE) Hafiz, Adamczyk e Johnson (2007)

Objetivo

O objetivo do Checkpointed System é estruturar um sistema de modo a possibilitar a recuperação de um estado conhecido válido caso um componente desse sistema venha a falhar.

Also Known As (AKA)

Snapshot Blakley e Heath (2004), Undo Blakley e Heath (2004)

Motivação

Uma falha de algum componente pode resultar em perdas ou danos a informações mantidas pelo mesmo. Sistemas que dependem de estados retidos para a operação correta devem ser capazes de se recuperar de danos ou perdas dessas informações

Aplicabilidade

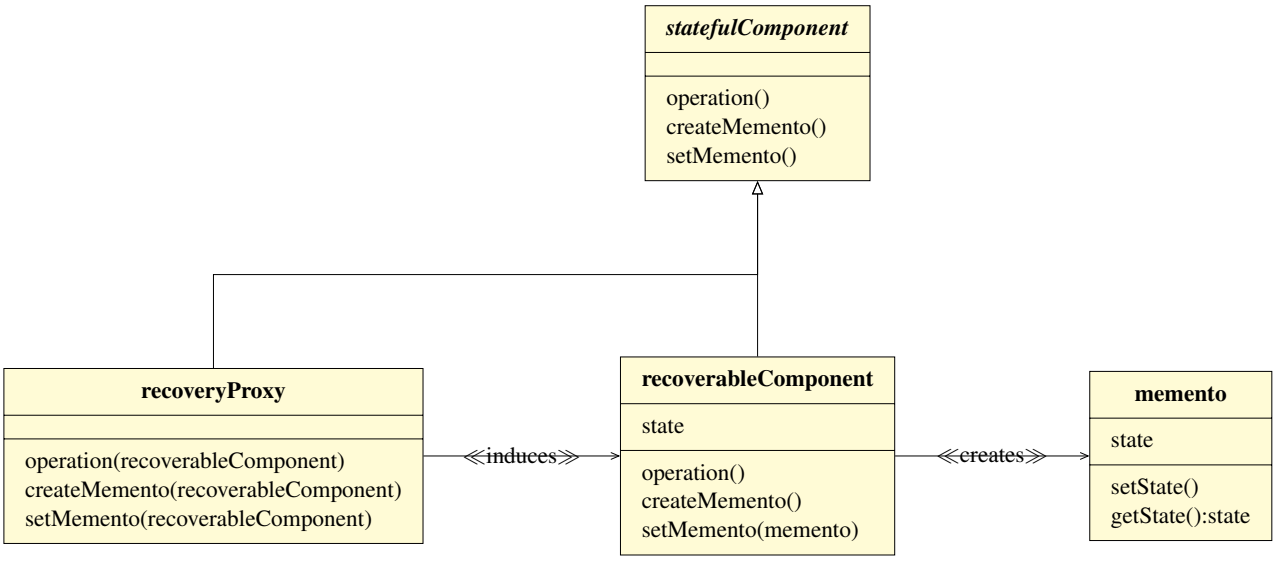
O uso do Checkpointed System é indicado quando:

- Operações em um componente atualiza seu estado
- Funcionamento correto do sistema depende da exatidão do estado dos seus componentes
- Falhas dos seus componentes podem causar perda ou corrompimento do estado de um componente
- Transações que ocorreram entre o momento do snapshot (checkpoint) e o momento em que o sistema teve que sofrer o rollback são irrelevantes, inconsequentes ou podem ser realizadas novamente.

Estrutura

O padrão Checkpointed System consiste em um Proxy de Recuperação (de acordo com Gamma et al. (1994), o proxy fornece um substituto/placeholder de outro objeto para, assim, controlar o acesso a esse objeto) e um Componente recuperável que periodicamente salva uma versÃo recuperável do seu estado na forma de um Memento (captura e externaliza, sem violar o encapsulamento, o estado interno de um objeto para que esse objeto possa ser retornado a esse estado mais tarde). Esse Memento pode ser usado para recuperar o estado do componente quando necessário. O Diagrama de Classes dO padrão Checkpointed System está representado na Figura 5.

Figura 5 – Diagrama de Classes do padrão Checkpointed System



Fonte: Adaptado de Blakley e Heath (2004)

Participantes

- Componente que apresente estado

  Classe abstrata. Define as operações desse componente

- Proxy de Recuperação

É um Proxy para componentes recuperáveis. Um Componente que apresenta estado. ”Cuidador” dos Mementos desse componente. Dá início à criação dos mementos quando o estado do Componente Recuperável é modificado. Detecta falhas e inicia a recuperação do estado ao induzir o Componente Recuperável a resgatar seu estado do Memento.

- Componente Recuperável

É um componente de estados. Implementa operações do componente. SaLva, periodicamente, o estado do componente em um Memento para permitir operações de recuperação mais tarde. Restaura o estado do componente quando requisitado.

- Memento

  É o estado externalizado do Componente Recuperável

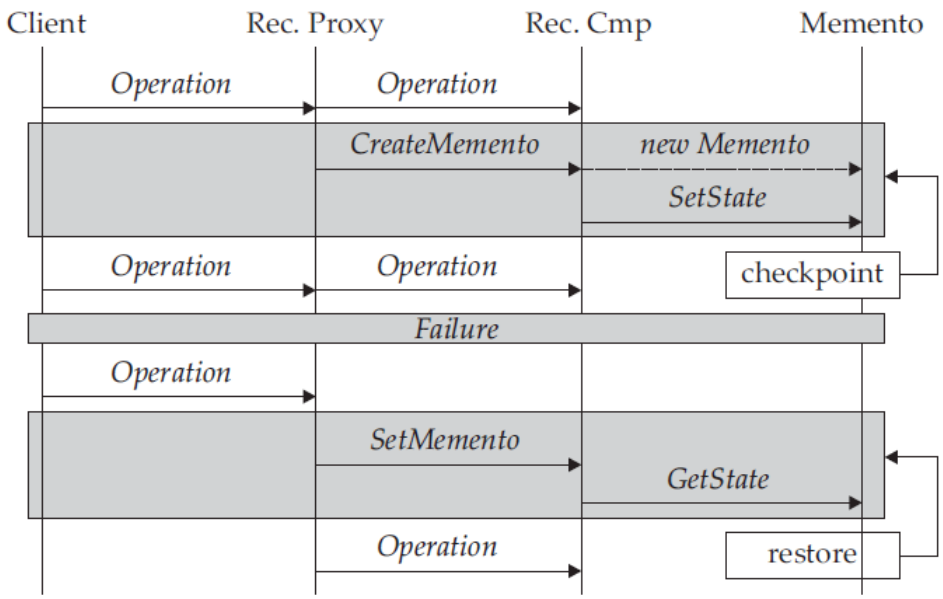
Colaborações

O funcionamento do Checkpointed System ocorre, basicamente, em trÊs etapas:

1. O Proxy de Recuperação responde a requisições para realizar operações;
2. O Proxy de Recuperação periodicamente induz o Componente Recuperável a criar um novo Memento para salvar seu estado atual;
3. Em caso de falha, o Proxy de Recuperação faz com que o Componente Recuperável restaure seu estado usando as informações armazenadas no Memento e, então, indica a realização das operações requisitadas pelo Componente Recuperável. Vale ressaltar que qualquer estado resultante de operações realizadas após o Memento mais recente será perdido.

O diagrama de sequência do padrão Checkpointed System está representado pela Figura 6

Figura 6 – Diagrama de Sequência do Checkpointed System



Fonte: Blakley e Heath (2004)

Consequências

O Uso do Checkpointed System:

- Aumenta a tolerância a falhas de componentes
- Aumenta a recuperação de erros de componentes

Por outro lado, seu uso também pode:

- Aumentar a utilização de recursos (para armazenar os Mementos)
- Aumentar a complexidade do sistema. A criação dos mementos pode depender da criação de filas de trabalho ou outro tipo de gerenciamento de transações para garantir a consistência dos estados neles armazenados.
- Aumentar a latência do sistema ou diminuir sua taxa de transferência caso a criação do Memento dependa de outros processos serem parados ou pausados.
- Permitir a perda de uma pequena quantidade de transações e seus estados associados
- Aumentar o preço do sistema por unidade de funcionamento

Implementação

Existe uma grande variadade de implementações para esse padrão. Muitos exemplos incluem uma grande variedade de configurações que permitem que o sistema reinicie a partir de um estado válido conhecido, esteja ele na mesma plataforma ou não.

Exemplo

Declaração da classe abstrata. Declara as operações a ser implementadas.

```
2       abstract class statefulComponent {
3           abstract protected function operation($var);
4           abstract protected function createMemento($var);
5           abstract protected function setMemento($var);
6       }
10
```

Classe herdeira da statefulComponent. Possui um array para armazenar os mementos e funções que induzirão o recoverableComponent a realizar as operações. Também possui uma função para checar o estado de erro do componente. Em caso positivo, o induz a recuperar seu último estado válido a partir de um memento.

```
3     class recoveryProxy extends statefulComponent {
4   var $mementos = array();

6   function operation($componente){
7     $componente->operation();
8   }

10  function createMemento($componente){
11    array_push($this->mementos, $componente->createMemento());
12  }

14  function setMemento($componente){
15    $componente->setMemento(end($this->mementos));
16  }

18  function detectaErro($componente){
19    if ($componente->erro === true){
20      $this->setMemento($componente);
21    }else{
22      return;
23    }
24  }
25 }

27
```

Classe stateful component. Possui um estado e estado de erro. Implementa a função operation() da classe pai. Não requer parâmetro. Função imprime uma mensagem na tela e altera o estado interno do componente. Também possui as funções createMemento() e setMemento(memento) que, respectivamente, retorna um memento com seu estado interno e recupera seu estado interno a partir de um memento.

```
1        class recoverableComponent extends statefulComponent{

3       var $estadoDefault = 0;
4       var $estado = 0;
5       var $erro = false;

7       function operation($var = NULL){
8         $this->estado++;
9         echo "Essa eh a operacao do recoverableComponent. Estado do
    componente: $this->estado";
10        echo "<br>";
11      }

13      function createMemento($var = NULL){
14        $memento = new memento();
15        $memento->setState($this);
16        echo "Memento de estado ".$memento->getState()." criado";
17        echo "<br>";
18        return $memento;
19      }

21      function setMemento($memento){
22        if($memento === false){
23          echo "Ocorreu um erro, mas nao ha nenhum memento anterior a ser
    recuperado";
24          $this->estado = $this->estadoDefault;
25          $this->erro = false;
26          return;
27        }
28        $this->estado = $memento->getState();
29        echo "<br>";
30        echo ">>>>>>>>>> Ocorreu um erro <<<<<<<<<<br> Estado $this->
    estado recuperado.";
31        $this->erro = false;
32      }
33    }


36
```

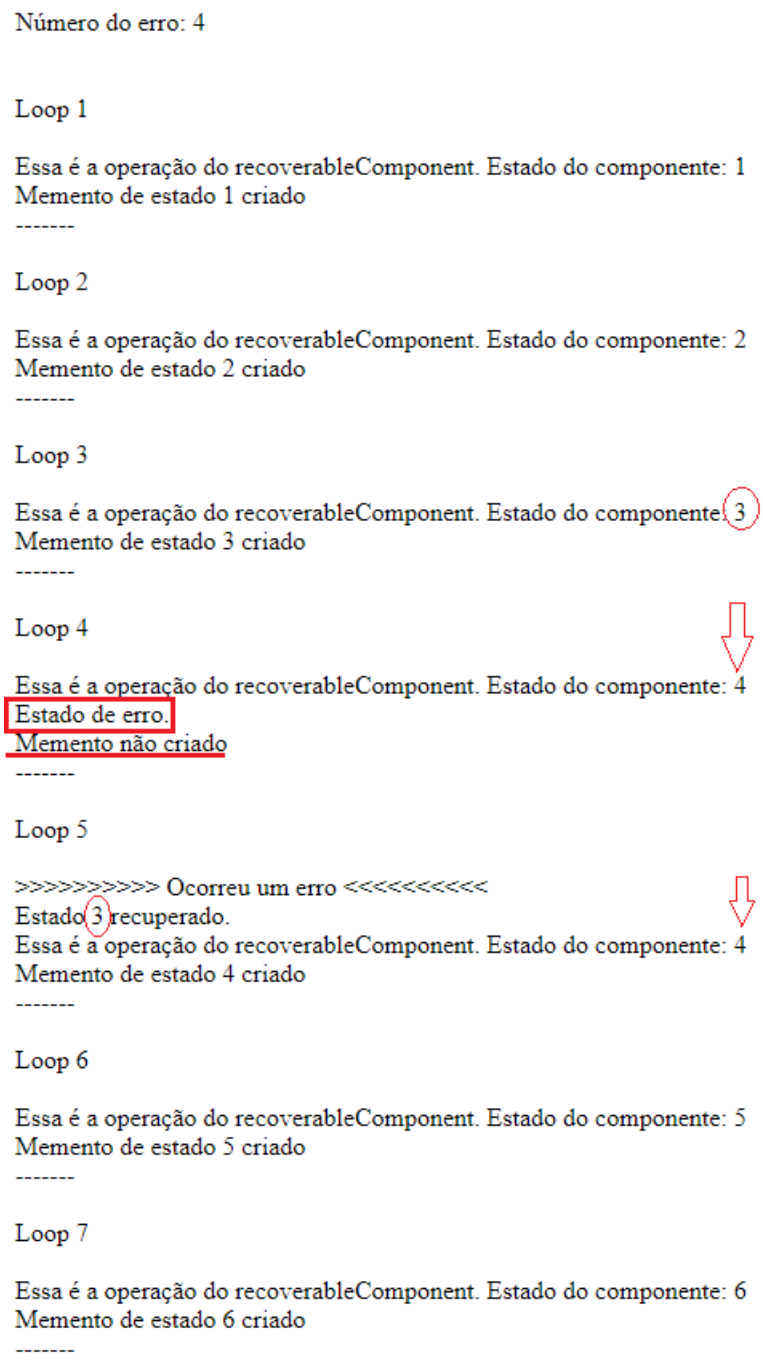
Classe Memento. Armazena um estado e possui as funcoes get() e set() desse estado

```
        class memento{
    var $estado;

    function getState(){
        return $this->estado;
    }

    function setState($component){
        $this->estado = $component->estado;
    }
}



```

A função checkpointedSystem(recoverableComponent, recoveryProxy) simula as requisições de um cliente (representadas pelos loop). O output desse código é representado pela Figura 7.

```
        function checkpointedSystem($recoverableComponent, $recoveryProxy){
    $erro = rand(1, 5);
    echo "Numero do erro: $erro";
    echo "<br>";
    echo "<br>";
    echo "<br>";
    for ($i=0; $i <= 6; $i++) {
        echo "Loop ".($i+1);
        echo "<br>";
        $recoveryProxy->detectaErro($recoverableComponent);
        echo "<br>";
        $recoveryProxy->operation($recoverableComponent);
        if(($i+1) === $erro){
            $recoverableComponent->erro=true;
            echo "Estado de erro. <br>Memento nao criado";
            echo "<br>";
            echo "-------";
            echo "<br>";
            echo "<br>";
        }else{
            $recoveryProxy->createMemento($recoverableComponent);
            echo "-------";
            echo "<br>";
        }
    }
}

```

Figura 7 – Output do Checkpointed System



Fonte: Gerada pelo autor

Usos conhecidos

A funcionalidade de salvar automaticamente a cada alteração (no Google Drive, por exemplo) é uma instanciação do padrão Checkpointed System

Padrões relacionados

O Recovery Proxy é um Proxy, descrito por Gamma et al. (1994) que armazena o estado do sistema em um Memento, também descrito por Gamma et al. (1994)

4.1.2 Replicated System

Tem como objetivo manter o sistema operante em caso de falha de um de seus componentes.

Nome e classificação do padrão

Replicated System

Available System Halkidis, Chatzigeorgiou e Stephanides (2004), Integrity and Availability Hafiz, Adamczyk e Johnson (2007), Exterior Security Hafiz, Adamczyk e Johnson (2007), Denial of Service (STRIDE) Hafiz, Adamczyk e Johnson (2007), Available Blakley e Heath (2004)

Objetivo

O objetivo do Replicated System é fornecer o serviço a partir de múltiplas fontes, possibilitando a recuperação em caso de falha de um ou mais componentes ou links.

Also Known As (AKA)

Redundant Components Blakley e Heath (2004), Horizontal Scalability Blakley e Heath (2004)

Motivação

Sistemas Transacionais são, muitas vezes, suscetíveis a falhas de links e/ou protocolos de comunicação ou de algum outro elemento do sisTema que acarrete em indisponibilidade do mesmo. Assim, é importante garantir a disponibilidade dos serviços de transação nesses casos.

Aplicabilidade

O uso do Replicated System é recomendado quando:

- O estado de um sistema for atualizado através de uma série de transações individuais
- O estado completo e o resultado de cada transação dever ser refletido precisamente pelo estado do sistema.
- Serviços equivalentes devam ser fornecidos simultaneamente a partir de múltiplos "pontos de presença", que por sua vez devem depender e consistentemente atualizar o mesmo estado do sistema.
- Falhas na comunicação/links forem mais prováveis do que falhas de componentes

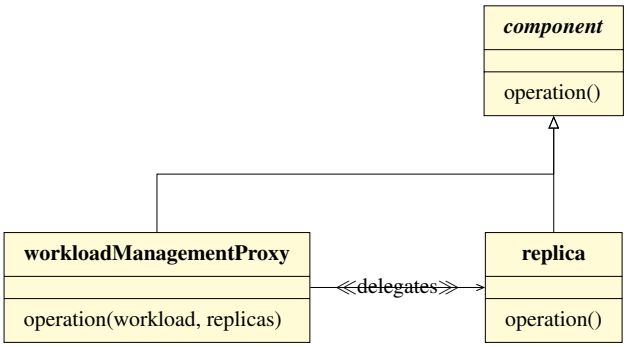
- Cada ponto de presença puder possuir acesso confiável a uma cópia mestre do estado do sistema.

- Os procedimentos operacionais exijam que um serviço seja realocado periodicamente de uma plataforma ou site para outro, e breves pausas no processamento para que essa realocação aconteça forem aceitáveis (a realocação pode ser necessária para adequar o ponto de fornecimento do serviço à localização que o utiliza, ou quando o serviço precisar ser realocado para uma plataforma mais capaz de atender altas demandas). O Serviço deve continuar sendo fornecido caso algum componente ou link falhe

Estrutura

O Replicated System consiste em duas ou mais Réplicas e um Proxy de Gerenciamento de Carga, que distribui a mesma entre os componentes. Todas as réplicas devem ser capazes de realizar o mesmo trabalho e podem ou não ser baseadas em estado. Caso sejam, a elas pode ser admitida alguma inconsistência. Caso as Replicas sejam baseadas em estado e precisem ser mantidas consistentes, o padrão Standby pode ser usado para garantir essa consistência de estados entre os componentes. O diagrama de classes desse padrão é representado pela Figura 8

Figura 8 – Diagrama de Classes do padrão Replicated System



Fonte: Adaptado de Yoder e Barcalow (1997)

Participantes

O padrão Replicated System conta com 3 participantes principais: componente, réplica e proxy de gerenciamento de carga.

- Component

É a classe abstrata que será herdada pelas duas outras classes. Garante que a função operation() será implementada pelas subclasses.

- Réplica

Implementa as operações. Todas as réplicas de um sistema replicado devem permitir o mesmo conjunto de operações

- Proxy de Gerenciamento de Carga

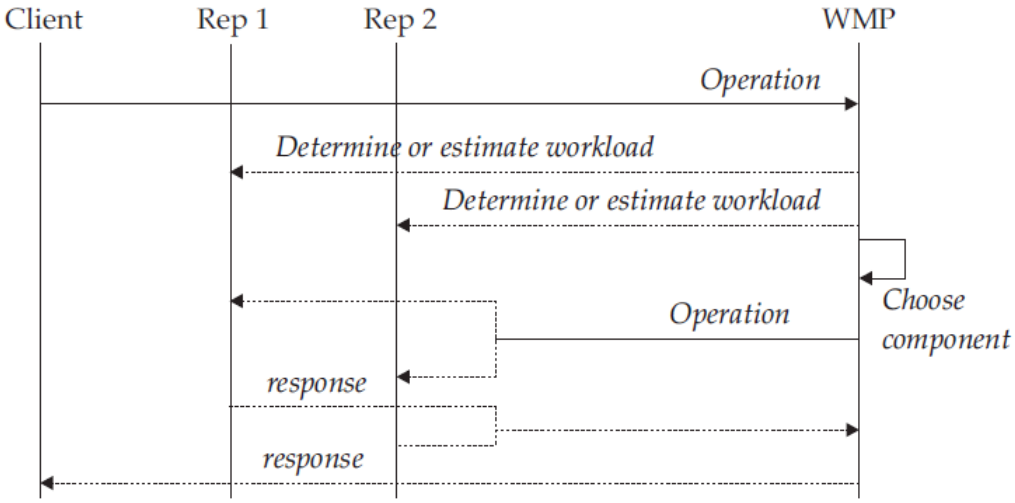
Delega operações a componentes baseado no algoritmo de planejamento de carga de trabalho

Colaborações

O funcionamento do Replicated System se dá em duas etapas básicas, representadas pela Figura 9:

1. O Proxy de Gerenciamento de Carga responde às requisições de operações
2. O Proxy de GeRenciamento de Carga delega essas requisições às réplicas mais adequadas a atendê-las

Figura 9 – Diagrama de Sequência do Replicated System



Fonte: Adaptada de Blakley e Heath (2004)

Consequências

O uso do Replicated System:

- Melhora a tolerância do sistema a falhas de componentes
- Melhora a habilidade do sistema de lidar com cargas distribuidas e falhas de comunicação

Por outro lado, seu uso também:

- Torna o Proxy de Gerenciamento de Carga um único ponto de falha e pode fazer com que os dados persistentes armazenem um único ponto de falha

Exemplo

Declaração da classe abstrata. Declara as operações a ser implementadas.

```
1   abstract class component {
2     abstract function operation($var1, $var2);
3   }
4
```

Classe réplica (representada por uma impressora. Contém 3 atributos e uma função. O erro é simulado pela variável codigoerro, que indica quantas operações podem ser feitas antes da falha.

```
1   class replica extends component {
2
3     var $estadoerro;
4     var $codigoerro;
5     var $nome;
6
7
8     public function __construct($nome, $min, $max){
9       $this->estadoerro = false;
10      $this->nome = $nome;
11      if($min === 0 && $max === 0){
12        $this->codigoerro = NULL;
13      }else{
14        $this->codigoerro = $erro = rand($min, $max);
15      }
16    }
17
18    public function operation($var1, $var2 = null ){
19      echo "Impressao do $var1 realizada pela $this->nome.";
20      echo "<br>";
21    }
22
23    public function erro(){
24      echo "-- A $this->nome encontrou um erro. O trabalho de impressao
    nao pode ser continuado por essa impressora. --";
25      echo "<br>";
26      $this->estadoerro = true;
27    }
28  }
29
30
```

Classe workloadManagementProxy. É o proxy de gerenciamento de carga, que tem uma função de delegar a carga. Recebe como parâmetro uma lista de trabalhos a serem feitos e uma lista de réplicas disponíveis para realizar esses trabalhos

```
1       class workloadManagementProxy extends component{

3     public function delega($componente, $carga){
4       $componente->operation($carga);
5     }

7     public function operation($impressoras, $fila){
8       $impressoraPadrao = current($impressoras);
9       $documento = current($fila);
10      echo "Impressora Padrao: $impressoraPadrao->nome";
11      echo "<br>";
12      $aux = 0;
13      while ($aux < sizeof($fila)){
14        if ($impressoraPadrao->estadoerro === false) {
15          if ($impressoraPadrao->codigoerro === $aux) {
16            $impressoraPadrao->erro();
17          }else{
18            $this->delega($impressoraPadrao, $documento);
19            $documento = next($fila);
20            $aux++;
21          }
22        }else{
23          $impressoraPadrao = next($impressoras);
24          echo "WMP diz: Impressora padrao alterada. Atual impressora
    padrao: $impressoraPadrao->nome";
25          echo "<br>";
26        }
27      }
28    }
29   }
30
```

A seguir, declara-se quais serão as réplicas, além da fila de documentos a serem impressos. O output do código está representado na Figura 10

```
$impressora1 = new replica("Impressora 1", 0, 10);
$impressora2 = new replica("Impressora 2", ($impressora1->codigoerro+1), 15);
$impressora3 = new replica("Impressora 3", ($impressora2->codigoerro+1), 20);
$impressora4 = new replica("Impressora 4", 0, 0);
$gerenciador = new workloadManagementProxy();

echo "Codigo de erro da $impressora1->nome: $impressora1->codigoerro";
echo "<br>";
echo "Codigo de erro da $impressora2->nome: $impressora2->codigoerro";
echo "<br>";
echo "Codigo de erro da $impressora3->nome: $impressora3->codigoerro";
echo "<br>";
echo "<br>";

$listaImpressoras = array($impressora1, $impressora2, $impressora3, $impressora4);


$filaDeImpressao = array();

for ($i=0; $i < 30; $i++) {
  $filaDeImpressao[$i] = "documento ".($i+1);
}

$gerenciador->operation($listaImpressoras, $filaDeImpressao);

```

Figura 10 – Output do padrão Replicated System

```
Código de erro da Impressora 1: 2
Código de erro da Impressora 2: 5
Código de erro da Impressora 3: 14

Impressora Padrão: Impressora 1
Impressão do documento 1 realizada pela Impressora 1.
Impressão do documento 2 realizada pela Impressora 1.
-- A Impressora 1 encontrou um erro. O trabalho de impressão não pode ser continuado por essa impressora. --
WMP diz: Impressora padrão alterada. Atual impressora padrão: Impressora 2
Impressão do documento 3 realizada pela Impressora 2.
Impressão do documento 4 realizada pela Impressora 2.
Impressão do documento 5 realizada pela Impressora 2.
-- A Impressora 2 encontrou um erro. O trabalho de impressão não pode ser continuado por essa impressora. --
WMP diz: Impressora padrão alterada. Atual impressora padrão: Impressora 3
Impressão do documento 6 realizada pela Impressora 3.
Impressão do documento 7 realizada pela Impressora 3.
Impressão do documento 8 realizada pela Impressora 3.
Impressão do documento 9 realizada pela Impressora 3.
Impressão do documento 10 realizada pela Impressora 3.
Impressão do documento 11 realizada pela Impressora 3.
Impressão do documento 12 realizada pela Impressora 3.
Impressão do documento 13 realizada pela Impressora 3.
Impressão do documento 14 realizada pela Impressora 3.
-- A Impressora 3 encontrou um erro. O trabalho de impressão não pode ser continuado por essa impressora. --
WMP diz: Impressora padrão alterada. Atual impressora padrão: Impressora 4
Impressão do documento 15 realizada pela Impressora 4.
Impressão do documento 16 realizada pela Impressora 4.
Impressão do documento 17 realizada pela Impressora 4.
Impressão do documento 18 realizada pela Impressora 4.
Impressão do documento 19 realizada pela Impressora 4.
Impressão do documento 20 realizada pela Impressora 4.
Impressão do documento 21 realizada pela Impressora 4.
Impressão do documento 22 realizada pela Impressora 4.
Impressão do documento 23 realizada pela Impressora 4.
Impressão do documento 24 realizada pela Impressora 4.
Impressão do documento 25 realizada pela Impressora 4.
Impressão do documento 26 realizada pela Impressora 4.
Impressão do documento 27 realizada pela Impressora 4.
Impressão do documento 28 realizada pela Impressora 4.
Impressão do documento 29 realizada pela Impressora 4.
Impressão do documento 30 realizada pela Impressora 4.
```

Fonte: Gerada pelo autor

Usos Conhecidos

Balanceadores de Carga de Rede (Web Servers de Frente Replicada, por exemplo) são exemplos do Replicated System Pattern.

Padrões Relacionados

O Replicated System pode usar o Standby para garantir a consistência dos estados entre as suas réplicas, caso seja necessário.

Utiliza um Proxy (Proxy de Gerenciamento de Carga), descrito por Gamma et al. (1994)

4.1.3 Discussão

Segundo Blakley e Heath (2004), os padrões Replicated e Checkpointed System se encaixam na cAtegoria de Available System Patterns, ou seja, padrões de projeto estruturais cujo objetivo é facilitar a construção de sistemas que provêm acesso ininterrupto e previsível aos serviços e recursos oferecidos aos usuários. Apesar de possuirem propósitos diferentes, são utilizados com o objetivo de minimizar os danos causados por eventuais falhas no sistema, garantindo a disponibilidade da informação nele contida. A disponibilidade da informação é um pilar da segurança da informação intimamente relacionada a questões operacionais das empresas e seus sistemas, e quaisquer déficits podem acarretar em danos e perdas muitas vezes irreparáveis.

O padrão Checkpointed System implementa, através de um proxy, um sistema de gerenciamento de informações dos componentes. Essas informações são armazenadas em Mementos, que funcionam como imagens de um componente em um determinado momento. A cada alteração de estado de um componente, o proxy o induz a criar um memento, armazenando nele o seu estado interno naquele determinado momento. Caso ocorra algum erro, o proxy induz o componente a recuperar seu estado a partir do último estado válido armazenado em um memento.

O Checkpointed System é composto por 4 classes:

- Stateful Component

A Stateful Component é uma classe abstrata que contém a função operation(), createMemento() e setMemento(), garantindo assim suas implementações nas classes filhas.

- Recovery Proxy

O Recovery Proxy é a classe que gerencia e encapsula todas as operações do Checkpointed System. Sua função operation() implementa uma chamada à função operation() do Recoverable Component. Além disso, também implementa as funções que induzem o Recoverable Component a realizar as suas próprias operações. Esse encapsulamento é importante para garantir um baixo acoplamento entre as classes. Outro ponto positivo de ter um proxy como responsável por realizar as chamadas dos métodos é minimizar qualquer erro ocorrido no sistema, uma vez que, caso esse erro esteja relacionado ao Recoverable Component em si, seu funcionamento pode ser prejudicado.

- Recoverable Component

A classe Recoverable Component é uma classe filha da classe Stateful Component. Sua implementação do método operation() representa a operação que altera o estado interno do componente. Idealmente, quando essa operação é realizada sem que haja erros, a operação createMemento() é utilizada para, como o próprio nome já diz, instanciar um objeto da classe

Memento que armazenará o estado interno do componente. Em caso de falha, o método SetMemento(memento) é utilizado para recuperar o estado interno de um componente em estado de erro a partir do último memento de estado válido (passado como parâmetro da função pelo Recovery Proxy), garantindo assim a disponibilidade de informações íntegras dentro do sistema.

- Memento

A quarta e última classe, Memento, armazena um estado internamente e possui métodos de get() e set() para esse estado.

Por sua vez, a classe Replicated System tem como objetivo garantir o funcionamento do sistema através de redundâncias. Essas redundâncias são implementadas através de réplicas de um componente, capazes de realizar o mesmo trabalho que esse componente. A presença dessas réplicas garante que, em caso de falha, o sistema continue funcionando (muitas vezes sem que o usuário sequer perceba). O gerenciamento das réplicas é realizado, assim como no Checkpointed System, por um proxy chamado de Workload Management Proxy (ou WMP). O WMP realiza a detecção de erros e, em caso de necessidade, delega a operação para alguma das réplicas.

O padrão Replicated System é composto por 3 classes:

- Component

É a classe abstrata que declara a função operation() que será implementada por suas classes filhas. Assim como no padrão Checkpointed System, a utilização de uma classe abstrata controla e garante que suas classes herdeiras irão implementar as funções necessárias, além de permitir uma maior escalabilidade do sistema.

- Replica

As réplicas são as instanciações do component que serão utilizadas pelo sistema em si. Sua implementação do método operation() (declarado como método abstrato na classe component) determina qual(is) será(ão) a(s) operação(ões) do objeto.

- Workload Management Proxy

Outra semelhança com o padrão Checkpointed System é a utilização de um proxy como o ponto central do funcionamento do sistema. Assim como no padrão anterior, o proxy é uma classe herdeira da classe component, e sua implementação do método operation() recebe como parâmetros as réplicas existentes, capazes de realizar o trabalho e representadas por impressoras prontas para imprimir documentos, e o trabalho a ser realizado (exemplificado através de um array de documentos a serem impressos). Basicamente, essa função realiza a delegação dos documentos entre as impressoras. Caso uma impressora encontre um erro, o WMP fará a detecção desse erro e continuará o trabalho de impressão utilizando a próxima réplica

disponível, até que a) todos os documentos tenham sido impressos; ou b) não haja mais réplicas disponíveis.

A escalabilidade que esse padrão fornece ao sistema é claramente visível quando altera-se o número de impressoras (réplicas) ou o número de documentos no array (carga). Independente de haver 3 ou 30 impressoras, o proxy será capaz de delegar e garantir o funcionamento contínuo do sistema em caso de erros, mesmo que o trabalho de impressão seja composto por 10, 100, 1000 (e assim por diante) documentos. Em outras palavras, através de alterações pontuais em apenas algumas linhas de código é possível adequar o sistema a praticamente qualquer demanda.

O proxy não é utilizado à toa por esses padrões de projeto. Em 1994, Gamma et al. (1994) apresentaram vários padrões de projeto, entre eles o proxy. Segundo os autores, o padrão fornece um placeholder para outro objeto com o objetivo de controlar o acesso a esse objeto. No trabalho em questão, o GoF apresentou diversas ocasiões em que o proxy pode ser aplicado, entre eles um proxy remoto (que fornece um representante local para um objeto em um endereço de espaço diferente), um proxy de proteção (que controla o acesso ao objeto original) e uma referência inteligente (que suBstitui a referência por ponteiros e realiza operações adicionais quando o objeto é acessado). Em todos os casos, o proxy deve implementar os mesmos métodos dos objetos que referencia, o que justifica a declaração de uma classe abstrata que garante essa implementação.

Vale ressaltar que a utilização de pontos centrais de funcionamento de um sistema representa um ponto de falha único, que pode resultar em gargalos indesejáveis. Como exemplo, caso o WMP do padrão Replicated System apresente algum defeito, todo o processo parará de funcionar, independente do número de réplicas disponíveis, que não funcionarão sem alguém para delegar os processos a elas. Cabe aos responsáveis pelo sistema a decisão de utilizar essas alternativas ou não, tendo em vista que, como sempre, abre-se mão de desempenho e custo em razão de segurança e disponibilidade.

4.2 Confidencialidade

São apresentados, a seguir, os padrões Authenticator e Authorization, considerados os mais relevantes do pilar Confidencialidade.

4.2.1 Authenticator

Tem como objetivo provar que aquela pessoa é, de fato, quem ela diz ser.

Nome e classificação do padrão

Authenticator
Protected System Halkidis, Chatzigeorgiou e Stephanides (2004) e Blakley e Heath (2004), Confidentiality Hafiz, Adamczyk e Johnson (2007) e Dangler (2013), Design Dangler (2013), Behavioral Bunke, Koschke e Sohr (2012), Authentication Bunke, Koschke e Sohr (2012), Perimeter security Hafiz, Adamczyk e Johnson (2007), Spoofing (Stride) Hafiz, Adamczyk e Johnson (2007), Access Control Röser (2012)

Objetivo

O padrão Authenticator apresenta um meio de conferir se o sujeito é quem ou o que ele afirma ser. O padrão realiza a autenticação do sujeito antes de fornecer acesso aos objetos

Motivação

Geralmente, um usuário é autenticado pelo sistema operacional quando loga pela primeira vez, mas pode ser que alguma autenticação adicional seja necessária ao acessar recursos sensíveis. Pode ocorrer de um impostor ganhar acesso aos recursos de um usuário legítimo (o que, quanto mais privilégios esse usuário tem, mais crítico se torna). Diferentes usuários podem ter diferentes níveis de acesso. Antes de permitir esse acesso, é necessário que o sistema identifique esse usuário.

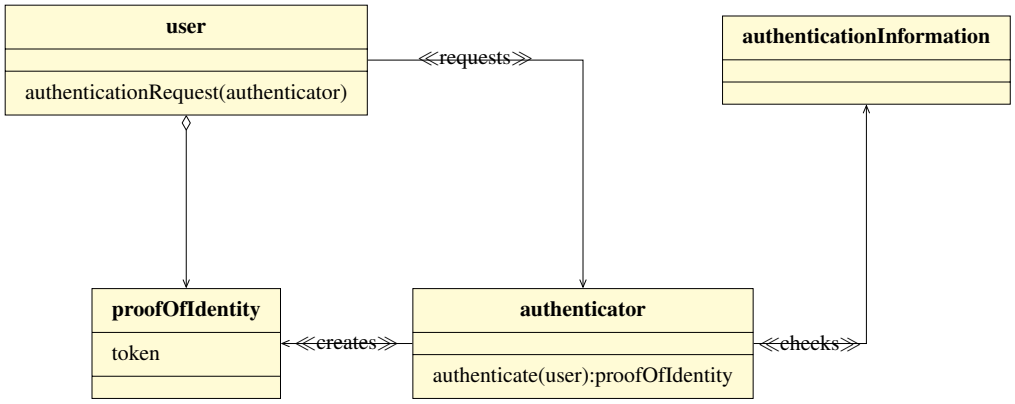
Aplicabilidade

O padrão Authenticator pode ser utilizado principalmente em sistemas distribuídos na forma de eventuais autenticações adicionais, de modo a verificar o acesso a objetos distribuídos em uma grande quantidade de acessos. Diferentes usuários podem utilizar diferentes modos de autenticação, portanto uma abordagem genérica é necessária. O processo de autenticação deve ser confiável, uma vez que outros sistemas confiam no Authenticator e baseiam suas decisões de acessos nele.

Estrutura

O Sujeito (geralmente o usuário), requisita acesso a recursos do sistema. O Authenticator recebe essa requisição e aplica um protocolo que utiliza alguma Informação de Autenticação. Se a autenticação é bem sucedida, o Authenticator cria uma Prova de Identidade, que pode ser explícita (um token, por exemplo), ou implícita. Seu diagrama de classes está representAdo na Figura 11.

Figura 11 – Diagrama de Classes do padrão Authenticator



Fonte: Adaptado de Dangler (2013)

Participantes

O padrão Authenticator é composto por 4 classes:

- User

Define a classe do usuário, com seus métodos e atributos. Um desses métodos é o authenticationRequest(authenticator), que cria uma chamada para que o Authenticator faça a autenticação desse usuário. O retorno desse método é o valor do atributo proofOfIdentity, explicado a seguir

- ProofOfIdentity

É o objeto que indica se o usuário é quem ele diz ser ou não. Pode ser implementada de várias maneiras. Quando criado, é atribuído a um usuário, permitindo o acesso ao sistema ou não.

- AuthenticationInformation

É a classe que armazena todas as credenciais cadastradas no sistema. É utilizada pelo Authenticator para realizar a comparação dos dados fornecidos pelo usuário com os dados armazenados no sistema

- Authenticator

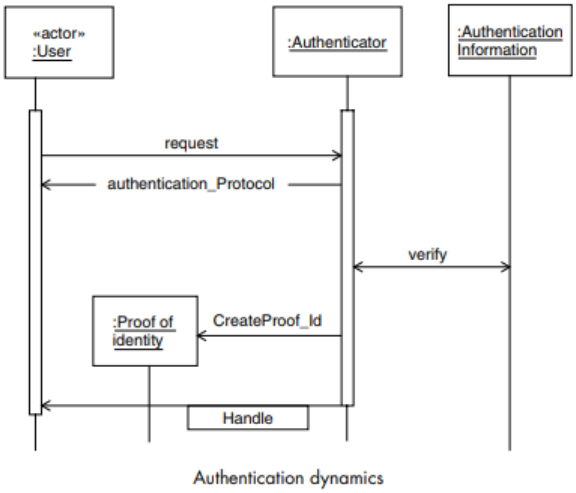
A classe authenticator é a classe que fará a autenticação do usuário. Possui a função authenticate que recebe como parâmetro um usuário que quer ser autenticado e compara suas credenciais com as credenciais armazenadas no sistema, retornando um objeto ProofOfIdentity, que será atribuído ao usuário.

Colaborações

O funcionamento do padrão Authenticator ocorre em três etapas, e é representado pela Figura 12:

1. O Usuário requisita o acesso a determinado recurso do sistema.
2. O Sistema requisita prova de que o usuário é quem ele diz que é
3. Essa prova é a Prova de Identidade, criada pelo Authenticator e atribuída ao usuário

Figura 12 – Diagrama de Sequência do Authenticator



Fonte: Schumacher et al. (2005)

Consequências

O uso do Authenticator tem como benefícios:

- Dependendo do protocolo e da informação de autenticação utilizada, é possível lidar com qualquer tipo de usuário, autenticando-os de múltiplas maneiras
- Como a informação de Authenticação é separada, pode-se armazená-la em uma área protegida (cujo acesso aos sujeitos será, no máximo, read-only)

- Pode-se utilizar vários algoritmos e protocolos, com variados níveis de força, para a autenticação. A seleção depende da segurança que se quer atingir, e o preço que se quer pagar por isso.
- A autenticação pode ser realizada em ambientes centralizados ou distribuídos
- É possível produzir uma prova de identidade a ser usada no lugar de autenticação adicional, o que aumenta a performance

Por outro lado:

- O processo de autenticação demora mais
- A complexidade e custo do sistema aumenta proporcionalmente ao nível de segurança.

Implementação

A implementação pode ser baseada em algo que o usuário sabe (como senhas), algo que o usuário possui (como sua identidade/cpf), algo que o usuário é (autenticação biométrica), onde o usuário está (terminal ou nó)

Exemplo

Define a classe de usuario, que possui 3 variáveis: login, senha e Prova de Autenticação. Além disso, possui uma função que requisita a autenticação a um authenticator.

```
2         class user{
3       var $login;
4       var $senha;
5       var $proofOfIdentity;
6
7       public function __construct($p1, $p2){
8         $this->login = $p1;
9         $this->senha = $p2;
10      }
11
12      function authenticationRequest($authenticator){
13        $this->proofOfIdentity = $authenticator->authenticate($this);
14      }
15    }
16
17
```

Classe proofOfIdentity, que é a classe cujo objeto será atribuído ao usuário que fizer a requisição de autenticação

```
    class proofOfIdentity{
  var $token;

  public function __construct($p){
    $this->token = $p;
  }
}

```

A classe usersInfo foi criada para que seus objetos armazenem o login e senha dos usuarios. Cada instanciação equivale a uma linha na tabela do banco de dados

```
    class usersInfo{
  var $login;
  var $senha;

  public function __construct($l, $s){
    $this->login = $l;
    $this->senha = $s;

  }
}

```

A classe authenticationInformation é a classe que armazena todos os usersInfo em um array. Um objeto dessa classe é passado como parâmetro do construtor do authenticator para ser utilizado na função authenticate(user)

```
    class authenticationInformation{
  var $identitiesArray = array();

  function addCredencial($login, $senha){
    $aux = new usersInfo($login, $senha);
    array_push($this->identitiesArray, $aux);
  }
}


```

Classe authenticator. Seu construtor recebe como parâmetros um objeto do tipo authenticationInformation (basicamente um array com várias credenciais) e possui a função authenticate() que recebe como parâmetro um usuário e compara as credenciais desse usuário (fornecidas via formulário) com as credenciais armazenadas no array usrinfo (credenciais corretas).

```
2     class authenticator{
3      var $usrinfo;
4      public function __construct($userinfo){
5        $this->usrinfo = $userinfo;
6      }
9      function authenticate ($user){
10       for ($i=0; $i < sizeof($this->usrinfo->identitiesArray); $i++) {
11         if ($user->login === $this->usrinfo->identitiesArray[$i]->login
   && $user->senha === $this->usrinfo->identitiesArray[$i]->senha) {
12           $resultado = true;
13           $prova = new proofOfIdentity($resultado);
14           echo "O áusurio esta AUTENTICADO.";
15           return $prova;
16         }else{
17           continue;
18         }
20       }
21       $resultado = false;
22       $prova = new proofOfIdentity($resultado);
23       echo "Ocorreu um ERRO NA ERRO NA AUTENTICACAO.";
24       return $prova;
25     }
26    }
27
```

Usos Conhecidos

Existem diversos serviços que utilizam a verificação em duas etapas (através do celular ou e-mail) de modo a aumentar a segurança. O uso de Chaves Públicas e Chaves Privadas também é um exemplo da utilização do Authorization.

Padrões Relacionados

Segundo Schumacher et al. (2005):

- O padrão Distributed Authenticator discute uma abordagem ao Authenticator em sistemas distribuídos.

- O padrão Single Access Point é um padrão abstrato que tem o Authenticator como uma aplicação concreta de si.
- O padrão Single Sign On é uma variação implementada em diversos sistemas
- O padrão Remote Authenticator (e Authorizer) é planejado para o acesso remoto a recursos compartiLhados em um sistema distribuído. Senhas são um protocolo específico de autenticação.

4.2.2 Authorization

Tem como objetivo definir níveis de acesso diferentes para cada usuário do sistema.

Nome e classificação do padrão

Authorization

Confidentiality Hafiz, Adamczyk e Johnson (2007) e Dangler (2013), Design Dangler (2013), Generic Concept Bunke, Koschke e Sohr (2012), Perimeter Security Hafiz, Adamczyk e Johnson (2007), Information Disclosure (STRIDE) Hafiz, Adamczyk e Johnson (2007), Access Control Röser (2012)

Objetivo

O padrão Authorization descreve quem é autorizado a acessar recursos específicos em um sistema, em um ambiente em que há recursos cujo acesso necessita ser controlado. O padrão indica, para cada entidade ativa que pode acessar os recursos, quais recursos ela pode acessar e como ela pode acessá-los

Motivação

Nem todo recurso em um sistema de informação deve ser acessível a todo sujeito (que pode ser um usuário ou processo). Uma necessidade é, antes de mais nada, definir como os recursos podem ser acessados. Caso contrário, um sujeito pode burlar facilmente as medidas de autorização.

- A estrutura do padrão Authorization deve ser independente do tipo de recurso. Para exemplificar, ele deve descrever o acesso pelos usuários a entidades conceituais, o acesso pelos programas a recursos do sistema operacional, e assim por diante, de uma maneira uniforme.
- A estrutura deve ser flexível o suficiente para acomodar diferentes tipos de sujeitos, objetos e permissões.
- Deve permitir, de um jeito relativamente fácil, a modificação dos direitos de um sujeito em resposta a mudanças em seus deveres e responsabilidades

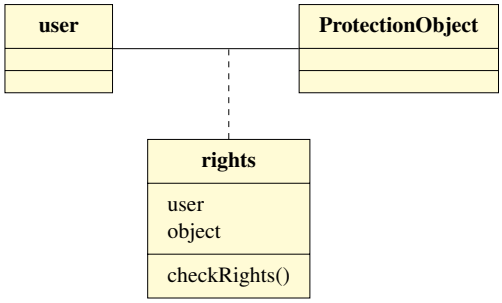
Aplicabilidade

Qualquer ambiente em que existem recursos cujo acesso precisa ser controlado. Seu uso evita que sujeitos não autorizados tenham acesso a recursos do sistema que não deveriam ser acessíveis ao mesmo.

Estrutura

A estrutura do padrão Authorization pode ser representada através de duas classes principais, além de uma terceira classe associativa entre os dois. Essa estrutura é apresentada pela Figura 13

Figura 13 – Diagrama de Classes do padrão Authorization



Fonte: Adaptado de Schumacher et al. (2005)

Participantes

Existem 3 classes principais que compõem o padrão Authorization

- Classe User

  Entidade que quer acessar um recurso

- Classe ProtectionObject

  Representa esse recurso

- Classe Rights

Classe associativa que indica qual o tipo de acesso que User tem sobre ProtectionObject

Colaborações

A classe User descreve uma entidade ativa que tenta acessar um recurso (Protection Object) de algum jeito. A classe ProtectionObject representa o recurso a ser protegido. A associação entre o sujeito e o objeto define a autorização (de onde o padrão tem seu nome). A classe associativa Rights descreve o tipo de acesso que o sujeito pode ter em relação àquele objeto. Através dessa classe, é possível conferir as permissões que um sujeito tem em relação a um objeto, ou ainda a quem é permitido o acesso a dado objeto. No exemplo contido neste trabalho, os tipos/níveis de acesso foram representados através de uma terceira classe: AccessType. Um objeto AccessType compõe a classe Right. Portanto, a classe RigHt armazena as informações sobre quem tem qual nível de acesso a qual objeto.

Consequências

O uso do padrão Authorization traz os seguintes benefícios:

- O padrão se aplica a qualquer tipo de recurso. Os Sujeitos podem ser processos em execução, usuários, papéis ou até mesmo grupos de usuário. Protections Objects podem ser transações, dados, áreas de memória, dispositivos de E/S, arquivos ou algum outro recurso. Tipos de acessos são definíveis individualmente e podem ser específicos de aplicação (em adição ao típico 'leitura/escrita')
- Adicionar ou remover autorizações é conveniente
- Alguns sistemas separam autorizações administrativas de autorizações de usuário para maior segurança, de acordo com o princípio de separação de deveres (Woo79 apud Schumacher et al. (2005))
- A requisição pode não precisar especificar o objeto específico em sua regra: o objeto pode ser implícito através de um objeto protegido existente (Fer75 apud Schumacher et al. (2005)). Sujeitos e tipos de acesso também podem ser implícitos, o que aumenta a flexibilidade (em troca de um tempo de processamento extra para que se deduza a regra necessária específica)

Por outro lado:

- Se existirem muitos usuários ou muitos objetos, um grande número de regras deve ser escrito.
- Pode se tornar difícil para o administrador de segurança perceber por que um determinado sujeito precisa de uma permissão, ou as implicações de uma nova regra.
- Definir regras de autorização não é o suficiente, uma vez que mecanismos de cumprimento dessa regra também são necessários

Implementação

Uma organização deve, de acordo com suas políticas, definir todos os acessos necessários aos recursos. A política mais comum é conhecida como "need-to-know", na qual entidades ativas recebem direitos de acesso de acordo com suas necessidades. De acordo com Schumacher et al. (2005), esse padrão é abstrato e existem muitas implementações. As duas mais comuns são Access Control Lists e Capabilities. A primeira se refere a listas que são armazenadas com os objetos para indicar quem é autorizado a acessá-lo, ao passo que a segunda se refere aos processos para definir suas permissões de execução. Além disso, os tipos de acessos devem ser orientados às aplicações.

Exemplo

Classe Protection object, que possui várias funções (representando os diferentes acessos possíveis em um objeto)

```
   class ProtectionObject
{
  function logaSistema(){
    echo "Logar no sistema<br>";
  }

  function acessaAgenda(){
    echo "Acessar a agenda<br>";
  }

  function editaCadastro(){
    echo "Editar os cadastros<br>";
  }

  function deletaTabela(){
    echo "Deletar tabelas<br>";
  }
}

```

Classe abstrata para roles. Contém uma função abstrata que será sobrescrita pelas classes herdeiras de acordo com as permissões dessas classes.

```
    abstract class accessType
    {
      var $rights;
      var $nome;
      abstract protected function roleFunction($object);
    }

```

Especializações da classe accessType. Cada role representa um nível de acesso, portanto cada role pode realizar funções específicas em relação ao ProtectionObject.

```
1    class roleVisitor extends accessType {


4   function __construct(){
5     $this->nome = "visitante";
6   }

8   public function roleFunction($object)
9   {
10    $object->logaSistema();
11  }

13  public function nome()
14  {
15    return "visitante";
16  }
17 }

19 class roleUser extends accessType
20 {

22  function __construct(){
23    $this->nome = "áusurio";
24  }

26  public function roleFunction($object)
27  {
28    $object->logaSistema();
29    $object->acessaAgenda();

31  }


34 }

36 class roleAdmin extends accessType
37 {

39  function __construct(){
40    $this->nome = "admin";
41  }

43  public function roleFunction($object)
44  {
45    $object->logaSistema();
```

```
46      $object->acessaAgenda();
47      $object->editaCadastro();

49    }

51  }

53  class roleMaster extends accessType
54  {

56    function __construct(){
57      $this->nome = "master";
58    }

60    public function roleFunction($object)
61    {
62      $object->logaSistema();
63      $object->acessaAgenda();
64      $object->editaCadastro();
65      $object->deletaTabela();
66    }

68  }


70
```

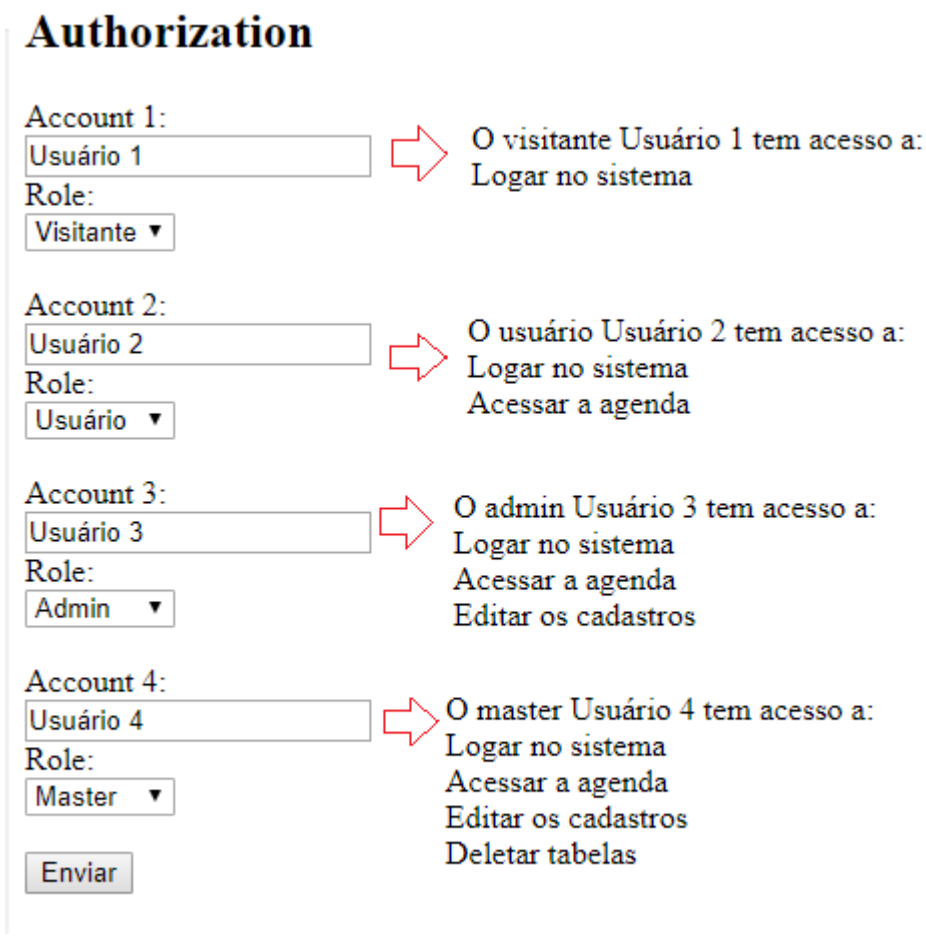
Classe usuário, que basicamente armazena suas credenciais.

```
1       class user
2   {
3     var $usrname = "";



6     public function __construct($p1)
7     {
8       $this->usrname = ($p1);
9     }
10  }


12
```

Classe associativa rights. Cada instanciação dessa classe armazena um usuário, um objeto e um nível de acesso. Essa classe vai definir qual usuário tem que tipo de acesso a determinado objeto. O output desse código está representado pela Figura 14.

```
    class rights
{
  var $user;
  var $accessType;
  var $object;

  function __construct($user, $role, $obj)
  {
    $this->user   = $user;
    $this->object = $obj;

    if ($role === "0") {
      $this->accessType = new roleVisitor();
    } elseif ($role === "1") {
      $this->accessType = new roleUser();
    } elseif ($role === "2") {
      $this->accessType = new roleAdmin();
    } elseif ($role === "3") {
      $this->accessType = new roleMaster();
    }

  }

  function checkRights()
  {
    echo "O " . $this->accessType->nome . " " . $this->user->usrname . "
   tem acesso a:";
    echo "<br>";
    $this->accessType->roleFunction($this->object);
  }
}



```

Figura 14 – Output do Padrão Authorization



Fonte: Gerada pelo autor

Usos Conhecidos

Existem inúmeros exemplos de como o Authorization pode ser utilizado. Por exemplo, um hospital pode permitir o acesso parcial às fichas dos pacientes aos enfermeiros, um acesso mais amplo aos médicos e o acesso total aos profissionais da TI. O controle de acesso é a base para o controle de acesso de diversos produtos comerciais, como o Unix, Windows, Oracle e muitos outros. Além disso, o Firewall é uma variação desse padrão que define o acesso a recursos de rede baseado no IP, por exemplo.

Padrões Relacionados

O padrão Role-Based Access Control é uma especialização desse padrão.

O padrão Reference Monitor complementa esse padrão definindo como aplicar as permissões definidas.

4.2.3 Discussão

Padrões como o Authorization e Authentication estão tão integrados na cultura de softwares que raramente nos recordamos que eles são, de fato, padrões de projeto. Atualmente, é inconcebível a fabricação de um software sem que haja um mínimo de controle de acesso aos seus dados e informações.

O padrão Authorization é voltado para aplicações em que diferentes usuários terão acesso a diferentes partes do sistema, ao passo que o Authentication é focado em qualquer sistema em que seja necessária a identificação do usuário. Apesar de existirem programas mais simples que não necessitem da implementação de níveis variados de acesso, é muito difícil encontrarmos sistemas em que o Authorization é utilizado, mas não o AuthenticatOr. Portanto, pode-se afirmar que ambos os padrões são encontrados na maioria dos sistemas atualmente.

O padrão Authenticator, apesar de representar um conceito relativamente simples, é essencial para a manutenção da segurança e da confidencialidade da informação de um sistema. Implementações do padrão que vão muito além de um simples token TRUE ou FALSE, representado no exemplo presente neste trabalho, podem tornar um sistema virtualmente impenetrável. Soluções que incluem várias etapas de autenticação tornam extremamente improvável que um invasor se passe por um usuário legítimo por meios digitais. Assim como foi descrito por Schumacher et al. (2005), quanto maior o grau de sofisticação da informação utilizada para gerar a prova de identidade, mais seguro é o sistema. Vale lembrar que, como sempre, o maior risco de invasão se encontra em técnicas de engenharia social, para as quais não existem tantos métodos de proteção.

Um detalhe importante da implementação desse padrão é a exteriorização da informação de autenticação do usuário em si. Essa exteriorização é reforçada pela classe authenticator, que é a única classe capaz de acessar tais informações, abstraindo o usuário de lidar com dados confidenciais. Outra vantagem dessa separação entre usuário e suas credenciais é a facilidade de manutenção do sistema como um todo. Caso alguma coisa necessite ser alterada, a classe do usuário em si (e, consequentemente, o sistema) dificilmente precisará ser alterada. Se essa alteração for feita na forma de validação, apenas o authenticator será alterado, já que é ele quem encapsula o algoritmo que utiliza a Informação de Autenticação para autenticar o usuário e criar a Prova de Identidade. Se a alteração for feita nos dados armazenados, apenas essa classe será alterada. O único papel do usuário é fornecer dados que serão comparados com os dados válidos armazenados pelo sistema.

O padrão Authorization, por sua vez, consiste basicamente em uma classe associativa que armazenará o usuário e o tipo de acesso que o mesmo tem em relação a um determinado objeto. A utilização dessa classe associativa corrobora o que foi dito por Dangler (2013): o padrão, por abstrair as permissões do usuário e do objeto, é flexível o suficiente para lidar com uma variedade de usuários, recursos e privilégios. Utilizando o Authorization, é simples realizar alterações de acesso sem que se afete o sistema inteiro.

No entanto, assim como nos padrões de disponibilidade apresentados anteriormente,

a utilização de padrões que apresentem entidades que centralizam alguns processos resultam em gargalos e pontos de falha únicos. Por outro lado, ao contrário dos padrões Checkpointed e Replicated System, a utilização dos padrões Authenticator e Authorization representam uma utilização consideravelmente menor de recursos do que os padrões de disponibilidade.

4.3 Integridade

Por fim, o último pilar a ser apresentado é o da integridade, e os padrões selecionados são o Input Validator e o Secure Access Layer.

4.3.1 Input Validator

O padrão input validator filtra tudo que entra no sistema de modo a evitar inconsistências que afetem a integridade do mesmo.

Nome e classificação do padrão

Input Validation

Integrity Bunke, Koschke e Sohr (2012) e Dangler (2013), Implementation Dougherty et al. (2009) e Dangler (2013), Structural Bunke, Koschke e Sohr (2012), Securing Applications Röser (2012)

Objetivo

O objetivo do padrão Input Validation é prevenir ataques e evitar que vulnerabilidades sejam exploradas através do input dos usuários. É necessário que o desenvolvedor identifique e valide todos os inputs externos de fontes de dados não confiáveis.

Motivação

O uso do input do usuário por uma aplicação é a origem de várias brechas de segurança, como ataques de buffer overflow, SQL injection e ataques de scripting cross-site. Dada a popularidade de aplicações com arquitetura cliente-servidor, existe a dúvida de se a validação do input deve ser realizada no lado do cliente ou no lado do servidor. Por um lado, a validação pelo servidor consome banda e recursos. Porém, a validação realizada apenas no lado do cliente também pode causar problemas sérios, uma vez que as validações client-side são inseguras. É fácil fazer um spoof de uma página de envio de formulários, burlando scripts na página original. Em todo caso, por mais que não se pode depender inteiramente da validação client-side, ela ainda é útil. Um feedback imediato do usuário pode evitar requisições ao servidor, economizando tempo e banda.

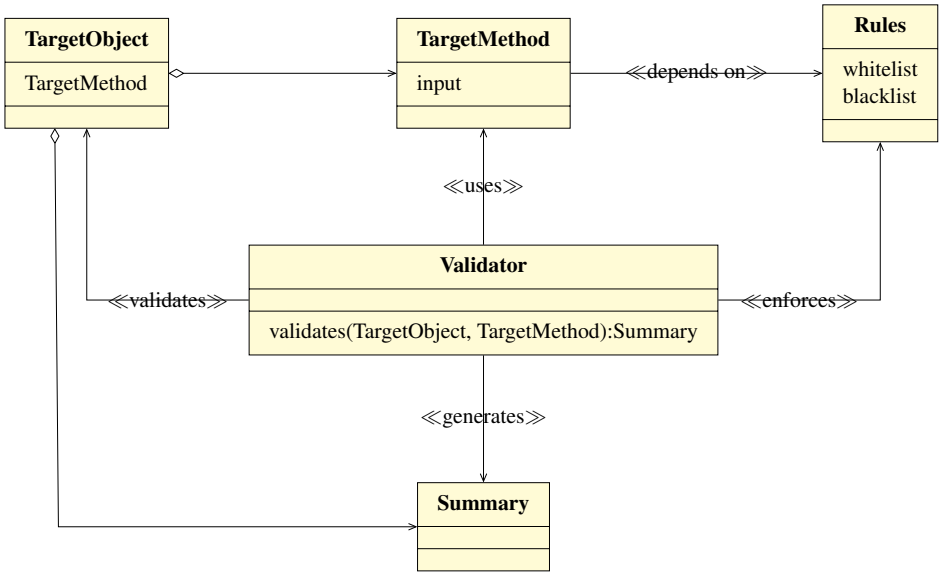
Aplicabilidade

O padrão Input Validation é aplicável a qualquer sistema que aceite dados de fontes não confiáveis. Quaisquer dados que cheguem à interface da aplicação vindos de fora dos seus limites de segurança requerem validação. Alguns exemplos incluem argv, ambientes, sockets, pipes, arquivos, signals, memória compartilhada e dispositivos em geral. Além disso, aplicações web apresentam fontes de input específicas para aplicações web (parâmetros GET e POST para formulários HTTP)

Estrutura

A estrutura do Input Validation requer a identificação e validação de cada um dos inputs. O resultado dessa validação será armazenado em forma de um sumário, atribuído ao TargetObject. Seu diagrama de classes está representado na Figura 15.

Figura 15 – Diagrama de Classes do Input Validator



Fonte: Adaptado de Netland, Espelid e Mughal (2006)

Participantes

Existem cinco elementos principais que participam da estrutura do Input Validator:

1. Target Object, que é o alvo do processo de validação e inclui propriedades sujeitas à inspeção
2. Target Method, que expõe propriedades do objeto
3. Regra, que define as constraints para as propriedades expostas do objeto
4. Validator, que controla o processo de validação, validando as propriedades expostas do objeto contra uma série de regras e reporta quaisquer violações dessas regras
5. Sumário de Validação, que fornece um sumário de todas as violações de regras

É possível, no entanto, simplificar e reduzir o número de participantes para 2:

1. Sistema, que aceita e valida os dados
2. Entidades externas fornecendo esses dados

Colaborações

Para explicar esse padrão, é possível fazer uma analogia com a segurança de um um aeroporto, por onde os passageiros devem passar por detectores de metal e máquinas de raio x. Nesse contexto, o Validator são os seguranças do aeroporto e seus equipamentos. Os Target Objects são os passageiros. Os Target Methods são os alvos individuais de inspeção (como malas, câmeras, mochilas e carteiras). As Regras são quais itens devem ser proibidos (como armas, drogas, etc). Por fim, os Sumários de Validação são os relatórios de incidentes, produzidos pelos profissionais do aeroporto em caso de violações de segurança.

Consequências

Utilizando o padrão de Input Validation:

- As mensagens-protocolo das aplicações são validadas e podem prevenir ataques.
- A abordagem White-List pode prevenir ataques utilizando vulnerabilidades ainda desconhecidas, contanto que esses ataques não correspondam aos inputs permitidos
- A detecção e responsabilização de tentativas de ataques podem ser ativados
- O Input Validation é separado da aplicação e fornece um ponto centralizado para todas as validações de input. Assim, o código de validação não é espalhado através da aplicação, o que o torna flexível a mudanças e de fácil manutenção

- Os validadores podem ser reutilizados por outras aplicações sem grandes modificações.

Por outro lado:

- A performance do sistema será impactada onde quer que o padrão seja utilizado
- Os input Validators podem apresentar novos pontos de falha que podem oferecer entradas ao sistema se exploradas.
- A complexidade do sistema aumenta, o que pode acarretar em aumento de custo de desenvolvimento e manutenção

Implementação

Para implementar o Input Validation, deve-se seguir 4 passos:

1. Identificar subsistemas potencialmente vulneráveis. Desenvolvedores devem separar componentes onde o conteúdo das mensagens será processado.
2. Determinar todas as fontes de input dos clients e definir quais serão consideradas inputs válidos. Netland, Espelid e Mughal (2006) recomenda a utilização de uma abordagem de whitelist, que especifica o formato de um input válido (também conhecido como Validação Positiva). Sendo assim, todo dado que não esteja conforme com o formato deverá ser descartado.
3. Configurar objetos e métodos alvo para os validadores usando o conjunto de fontes de input dos clients. Construir regras para definições válidas de input a partir do passo anterior e mapear essas regras de acordo com os métodos alvo correspondentes.
4. Implantar os validadores, que devem ser integrados em todas as aplicações web potencialmente vulneráveis.

Exemplo

Para exemplificar, foi implementado um sistema de checagem de bagagem em um aeroporto. A classe passageiro representa o targetObject e seus TargetMethods são representados pela classe bagagem, que armazena os itens levados pelo passageiro.

```
class passageiro{

  var $bagagem;
  public function __construct($p1){
    $this->bagagem=$p1;
  }
  var $sumario;
  var $inputPermitido;
}

class bagagem{
  var $itens = array();
  public function __construct($p1){
    $this->itens=$p1;
  }
}
```

As regras são implementadas pela classe aeroporto.

```
class aeroporto{
  var $regras = array();
  public function __construct($r1){
    $this->regras=$r1;
  }
}
```

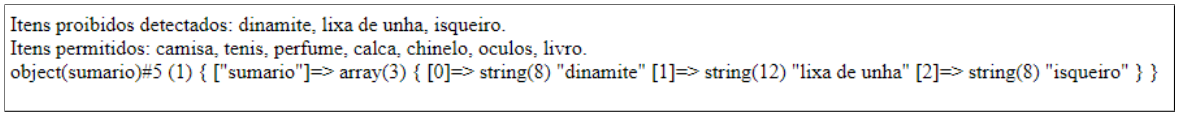
As duas últimas classes são a classe segurança, que vai inspecionar a bagagem do passageiro utilizando as regras do aeroporto, e o sumário, que será o resultado dessa inspeção (itens proibidos).

```
class sumario{
  var $sumario = array();
}

class seguranca{

  public function inspeciona($p, $a){
    $sumario = new sumario();
    for ($i=0; $i<sizeof($a->regras); $i++) {
      for ($j=0; $j<sizeof($p->bagagem->itens); $j++) {
        if ($p->bagagem->itens[$j] === $a->regras[$i]){
          array_push($sumario->sumario, $p->bagagem->itens[$j]);
        }else{
          continue;
        }
      }
    }
    $p->inputPermitido = array_diff($p->bagagem->itens, $sumario->sumario);
    echo "Itens proibidos detectados: ".implode(', ',$sumario->sumario).".<
   br>Itens permitidos: ".implode(', ',$p->inputPermitido).".";
    return $sumario;

  }
}
```

Por fim, são instanciados os elementos e o passageiro é inspecionado pelo segurança, que utiliza as regras do aeroporto e gera um sumário. O output está representado através da Figura 16.

```

$itens = array('camisa','tenis','perfume','isqueiro','lixa de unha','calca'
   ,'chinelo','oculos','livro','dinamite');
$blacklist = array('dinamite','arma','faca','lixa de unha','bomba','
   isqueiro');
$bagagem = new bagagem($itens);
$passageiro1 = new passageiro($bagagem);
$aeroporto = new aeroporto($blacklist);
$seguranca = new seguranca();

$passageiro1->sumario = $seguranca->inspeciona($passageiro1, $aeroporto);
echo "<br>";

var_dump($passageiro1->sumario);
```

Figura 16 – Output do Input Validator



Fonte: Gerada pelo autor

Usos Conhecidos

Existem incontáveis exemplos de uso do padrão input validator. A seguir, apresenta-se alguns exemplos:

- Commons Validator

Componente de validação open-source que originou do framework Apache Struts. Os objetos alvo são JavaBeans. Regras são chamadas de Validator Actions e são métodos estáticos booleanos em objetos Java. O componente permite que desenvolvedores configurem métodos alvo e regras nos arquivos de configuração.

- Stinger

Mecanismo de validação de requisições HTTP usado em um ambiente J2EE. A requisição HTTP é o objeto alvo. O mecanismo suporta regras de expressões regulares e regras para partes faltantes ou extras em requisições HTTP. Desenvolvedores podem especificar as regras utilizando a Linguagem de Descrição de Validação de Segurança, que é adaptado para mapear os métodos alvo em requisições HTTP.

- .NET Validation Server Controls

São parte dos formulários web no framework .NET e permitem que desenvolvedores executem a validação de inputs dos clientes em aplicações web. Os objetos alvo são os controles de input do servidor, e o framework fornece regras pré-definidas tais como campos obrigatórios, ranges e expressões regulares. Os controles de validação são especificados como parte da lógica de apresentação.

Padrões Relacionados

O padrão Client Input Filters discute o problema da validação de input em um contexto web, e foca nas melhores práticas e problemas na implementação de aplicações web.

O padrão Intercepting Validator é um padrão J2EE para validar input do cliente. Não está ligado à lógica de negócio, e tenta filtrar ataques conhecidos no início do processo de gerenciamento de requisições. Em contraste com o Input Validation, Netland, Espelid e Mughal (2006) recomenda a utilização do padrão Intercepting Validator para também proteger de ataques de injeção.

4.3.2 Secure Access Layer

O padrão Secure Access Layer atua como um mediador entre o sistema e outras aplicações, exteriores a ele, garantindo a integridade dos dados transportados entre eles.

Nome e classificação do padrão

Secure Access Layer

Integrity Bunke, Koschke e Sohr (2012) e Dangler (2013), Architectural Dangler (2013), Generic Concept Bunke, Koschke e Sohr (2012), Securing Applications Röser (2012)

Objetivo

Geralmente, a segurança de uma aplicação é construída ao redor de mecanismos de segurança já existentes do sistema operacional, de redes e banco de dados. Com base na afirmação de que "um sistema só é tão seguro quanto seu link mais fraco", o padrão Secure Access Layer descreve como implementar uma camada de acesso de segurança adicional para adicionar esses mecanismos de seguranças e manter a comunicação indo e vindo da aplicação segura.

Also Known As (AKA)

Using Low-level security Yoder e Barcalow (1997) e Röser (2012), Using Non-application security Yoder e Barcalow (1997) e Röser (2012), Only as strong as the weakest link Yoder e Barcalow (1997);

Motivação

Fazendo uma analogia com uma base militar, quando uma informação é transportada, é comum ela ser transportada encriptada, em um disco, dentro de uma maleta trancada e presa ao pulso do transportador com uma algema. Trazendo para a realidade de sistemas de informação, a integração com outros sistemas pode acarretar em vulnerabilidades. Portanto, a implementação de mecanismos de segurança que lide com essa troca de informações é necessária e importante para garantir a integridade dos dados em questão, uma vez que, se o padrão não for adotado, a checagem desses dados em cada etapa de integração pode se tornar custosa do ponto de vista de desenvolvimento.

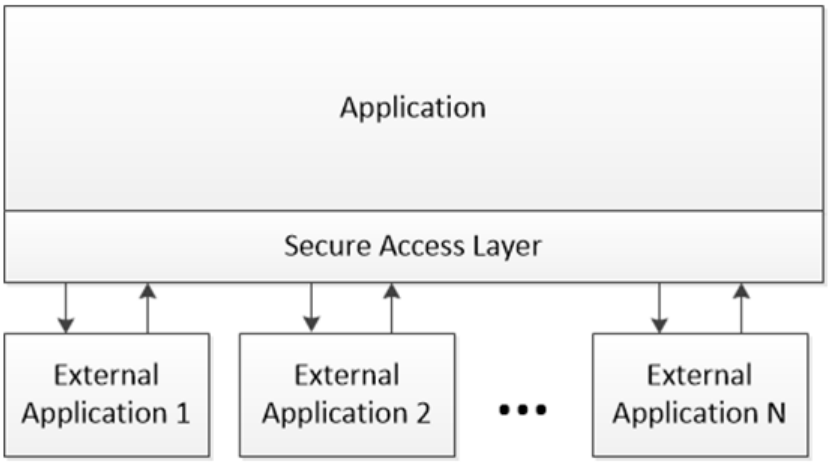
Aplicabilidade

Esse padrão deve ser utilizado em aplicações que se comunicam e são integradas a outros sistemas. Desse modo, a Secure Access Layer fornece uma interface para protocolos padrões de comunicação.

Estrutura

O padrão Secure Access Layer é implementado entre a aplicação e o meio externo, agindo como uma ponte de comunicação. Sua estrutura está representada pela Figura 17.

Figura 17 – Estrutura do padrão Secure Access Layer



Fonte: Dangler (2013)

Participantes

O padrão Secure Access Layer possui, basicamente, 3 participantes:

- Aplicação

  Se comunicará com o meio externo

- Secure Access Layer

  Responsável por mediar essa comunicação, adotando medidas de segurança integradas com os mecanismos de segurança das aplicações externas

- Aplicações externas

  Estão do outro lado da ponte, e se comunicarão com a aplicação através do Secure Access Layer

Colaborações

A aplicação precisa se comunicar com outro sistema, externo a ela. Entre ela e tal sistema, existe uma camada extra de segurança, a Secure Access Layer, que media essa comunicação, encapsulando os dados e isolando a aplicação dos sistemas externos, tornando a integração mais fácil e segura.

Consequências

Os benefícios do uso do padrão Secure Access Layer são:

- O isolamento da comunicação com fontes externas torna mais fácil a integração com novos componentes de segurança ou o upgrade dos componentes existentes.
- A portabilidade para outros sistemas é mais fácil, já que apenas essa camada de acesso seguro necessita de adaptação (a aplicação é independente)

Por outro lado:

- O desenvolvimento de uma Secure Access Layer que abrange um grande número de sistemas diferentes é difícil e pode acarretar em sobrecarga desnecessária.
- Adaptar uma Secure Access Layer a aplicações já existentes pode ser difícil, uma vez que o código para a comunicação segura pode estar espalhado através da aplicação já existente.

Implementação

O padrão Secure Access Layer é implementado através da construção de uma camada de baixo nível própria da aplicação, que encapsula toda a comunicação que vai e vem. Essa camada de segurança é construída em volta de mecanismos de segurança já existentes de SO, banco de dados e de rede. Se tal mecanismo não existir, é necessário construí-lo dentro do Secure Access Layer para garantir a segurança da comunicação que é transportada da e para a aplicação. É importante que essa camada isole o desenvolvedor da mudança para sistemas e mecanismos subjacentes.

Usos Conhecidos

O Secure Shell (SSH) inclui protocolos de segurança para a comunicação em sessões X11 e pode usar criptografia RSA através de conexões TCP/IP

SSL (Netscape Serve) fornece uma Secure Access Layer que clientes web podem utilizar para garantir comunicações seguras

O Oracle fornece sua própria Secure Access Layer que aplicações podem utilizar para se comunicar com ele.

CORBA Security Services especifica como autenticar, administrar, auditar e manter a segurança através de um sistema de objetos distribuído CORBA. Qualquer Secure Access Layer de uma aplicação CORBA irá se comunicar com o Security System CORBA.

O Caterpillar/Framework do Modelo Financeiro NCSA usa uma Secure Access Layer fornecida pelo LensSession no Smalltalk do VisualWorks do ParcPlace

O Programa de regiostro PLoP ’98 passa por uma Secure Layer para acessar o sistema. Primeiro, a aplicação roda sobre o SSL do servidor Apache. Além disso, todas as informações de cartões de créditos foram armazenadas encriptadas quando os usuários se inscreveram pro PLoP ’98

4.3.3 Discussão

Quando um sistema interage com meios externos, ele está vulnerável a diversos tipos de ameaças, falhas e, consequentemente, ataques. Os padrões de projeto vistos até aqui, principalmente os padrões de confidencialidade, visam garantir, também, a integridade do sistema. É por esse motivo que é tão difícil separar os padrões em apenas uma categoria bem definida. Além de procurarem solucionar problemas relacionados a vários aspectos da segurança, algumas vezes essa categorização é feita de maneira arbitrária pelos autores.

Os padrões selecionados para representar o pilar da Integridade foram o Input Validator e o Secure Access Layer. Ambos têm como foco lidar com fatores externos ao sistema. O Input Validator é responsável por validar dados advindos de fontes terceiras, ao passo que o Secure Access Layer fornece uma interface de comunicação entre o sistema em si e algum outro sistema externo.

O Input Validator é classificado por Bunke, Koschke e Sohr (2012) um padrão estrutural, ou seja, é um padrão que se preocupa com as estruturas ou composições implementadas em um sistema. O exemplo contido neste trabalho corrobora essa ideia: o padrão implementa uma estrutura (representada através de uma analogia utilizando um aeroporto) responsável por filtrar e barrar dados indesejados, impedindo sua entrada no sistema.

Existem diversas implementações desse padrão. As duas principais funcionam através da utilização de whitelists (proposto por Netland, Espelid e Mughal (2006)), ou de blacklists. As whitelists, conhecidas como validação positiva, determinam quais inputs são válidos para o sistema, descartando tudo que não estiver em conformidade com elas. As blacklists, por sua vez, especificam quais inputs não são válidos, permitindo todo o resto. A decisão de qual abordagem utilizar depende do modelo de negócio que está sendo implementado. Recomenda-se utilizar a abordagem que gerar a menor lista. Por exemplo: se, de 100 itens possíveis, 10 não são permitidos, mas os outros 90 são, usa-se a blacklist. Se a situação for inversa e, desses 100 itens, apenas 10 são permitidos, ao passo que os outros 90 são poibidos, usa-se a whitelist. Em resumo, é mais fácil e econômico realizar a filtragem quando o catálogo do que se procura é menor.

Em sistemas client-server, a implementação do padrão depende também de outro fator: onde realizar o tratamento dos dados recebidos. A implementação no servidor é mais segura, mas consome mais recursos, uma vez que cada validação resulta em dados trafegados entre o cliente e o servidor, consumindo banda e processamento. Por outro lado, a implementação no client side, apesar de economizar banda, pode ser facilmente burlada através de técnicas de spoofing ou, ainda mais simples, através da desabilitação do mecanismo de validação de formulários (a maioria dos navegadores oferecem ao usuário a opção de desabilitar componentes

do sistema, como o JavaScript).

Dos padrões apresentados, o mais difícil de ser exemplificado é o Secure Access Layer. Isso se dá por dois motivos principais: primeiro, o sistema necessita de um outro sistema, externo àquele que implementa o padrão, para ser utilizado; segundo: o padrão é considerado um conceito genérico por autores como Bunke, Koschke e Sohr (2012), que explica ainda que padrões na categoria conceito genérico não apresentam aspectos comportamentais e estrutura definida (o que, aliás, é o caso do padrão Memento, utilizado pelo Checkpointed System). Por esse motivo, o Secure Access Layer é o único padrão (dos 6 que esse artigo se propôs a apresentar) que não foi implementado.

Para explicar o funcionamento do Secure Access Layer, Yoder e Barcalow (1997) utiliza uma base militar como analogia. No exemplo, o autor cita a utilização de diversos mecanismos de segurança com o objetivo de minimizar os riscos advindos do transporte de informações entre duas localidades. Como, na maioria das vezes, um sistema deve se comunicar com o meio externo, o Secure Access Layer é utilizado para garantir que o transporte de dados não represente um risco à segurança do sistema.

A utilização de uma interface que centraliza e protege os dados que transitam entre o sistema e o ambiente externo apresenta vantagens importantes. Além da camada extra de segurança que o padrão fornece, a manutenção e adaptação desse sistema se tornam consideravelmente mais simples. O Secure Access Layer evita que o tratamento desses dados necessite ser feito individualmente em todos os pontos de entrada e saída do sistema. Além disso, a migração de um sistema (troca de SGBD, por exemplo) é menos complicada porque apenas essa camada deve ser adaptada.

Talvez o exemplo mais conhecido da utilização do Secure Access Layer é o SSL, que utiliza criptografia assimétrica para garantir que os dados que trafegam entre sistemas só possam ser acessados pelo sistema a qual eles se destinam. Basicamente, a criptografia assimétrica utiliza duas chaves para estabelecer uma comunicação segura. Cada entidade possui duas chaves: uma pública, que todos têm acesso, e uma privada, que apenas essa entidade tem acesso. Na prática, a entidade que envia a informação utiliza a chave pública da entidade de destino para encriptar essa informação. Essa criptografia só poderá ser desfeita através da chave privada da entidade de destino, garantindo a confidencialidade e integridade dessa informação.

Um dos motivos pelos quais o SSL se tornou tão difundido nos últimos anos foi a pressão exercida pelo Google para a adoção do protocolo (cada vez mais a empresa vem penalizando sites que não possuem o certificado), o que dá uma dimensão da importância do padrão.

5 CONCLUSÃO E TRABALHOS FUTUROS

A segurança da informação está em evidência e, com ela, todas as questões relacionadas à proteção de sistemas de informação. Na engenharia de software, um código mais simples, porém robusto e seguro é melhor do que um código grande que pode apresentar vulnerabilidades e brechas. Para alcançar essa simplicidade e robustez no sistema, é recomendada a utilização de padrões de projeto, que são soluções já testadas e consolidadas para problemas recorrentes encontrados por desenvolvedores em novos sistemas.

A utilização de padrões de projeto tira do desenvolvedor a responsabilidade de reinventar a roda e buscar criar uma solução do zero, permitindo-o focar em outros aspectos do sistema. No entanto, o cenário dos padrões de projeto voltados à segurança da informação não é tão amigável quanto parece, principalmente dada a importância que o papel da segurança tem em uma aplicação.

Através da revisão da literatura, constatou-se que existem muitos padrões, insuficientes informações sobre eles e, como se não bastasse, não há um método padronizado de categorização e organização desses padrões. Com o objetivo de apresentar os padrões de projeto mais relevantes na área de segurança da informação, foram feitas duas análises: dos padrões em si e dos meios de organização e categorização desses padrões.

Após o cruzamento dos resultados dessas duas análises, 6 padrões foram selecionados, representando, em pares, cada um dos 3 pilares da segurança da informação: Input Validator e Secure Access Layer representando a Integridade, Checkpointed e Replicated System representando a Disponibilidade e, por fim, Authenticator e Authorization representando a Confidencialidade.

Utilizando o template para apresentação de padrões de projeto apresentado por Gamma et al. (1994), os 6 padrões foram apresentados e discutidos, levando em conta sua aplicação, vantagens e desvantagens. Além disso, com exceção do padrão Secure Access Layer, os padrões selecionados foram implementados em PHP de modo a fornecer um exemplo real e, consequentemente, facilitar o entendimento desses padrões.

Os resultados apresentados por esse trabalho são apenas uma pequena amostra do poder e eficiência dos padrões de projeto de segurança existentes. No entanto, apesar de abrangerem diferentes etapas de funcionamento de um sistema, a apresentação de apenas 6 padrões é um bom começo, mas não é o suficiente para cobrir todos os problemas de segurança existentes que podem ser solucionados por ferramentas já existentes e comprovadas. Para colocar em perspectiva, menos de 3% dos padrões levantados pela pesquisa foram apresentados, o que significa que muitos padrões essenciais ao desenvolvimento de software ficaram de fora. Trabalhos futuros podem focar nesse ponto para ajudar a construção de um catálogo cada vez mais completo e confiável dos padrões de projeto de segurança da informação.

BIBLIOGRAFIA

BLAKLEY, Bob; HEATH, Craig. *Security Design Patterns - Technical Guide*. 2004.

BUNKE, Michaela; KOSCHKE, Rainer; SOHR, Karsten. Organizing Security Patterns Related to Security and Pattern Recognition Requirements. *International Journal on Advances in Security, vol 5 no 1 & 2*, 2012.

CHRISTOPHER ALEXANDER Sara Ishikawa, Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction (Cess Center for Environmental)*. Oxford University Press, 1977. ISBN: 0195019199,9780195019193. Disponível em: <<http://gen.lib.rus.ec/book/index.php?md5=6A09E611680C7FA35B6C06824962A9A1>>.

DANGLER, Jeremiah Y. *Categorization of Security Design Patterns*. 2013. Tese (Doutorado) – East Tennessee State University.

DOUGHERTY, Chad et al. *Secure Design Patterns*. 2009.

ELDER, Darrell M. Kienzle Matthew C. *Security Patterns for Web Application Development*. 2001.

GAMMA, Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Edição de Pearson Education. Pearson Education (US), 1994.

HAFIZ, Munawar; ADAMCZYK, Paul; JOHNSON, Ralph E. Growing a pattern language (for security). In: *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software - Onward! '12*. ACM Press, 2012. DOI: <10.1145/2384592.2384607>.

HAFIZ, Munawar; ADAMCZYK, Paul; JOHNSON, Ralph E. Organizing Security Patterns. *IEEE Computer Society*, 2007.

HALKIDIS, Spyros T.; CHATZIGEORGIOU, Alexander; STEPHANIDES, George. A Qualitative Evaluation of Security Patterns. *ICICS 2004, LNCS 3269, pp. 132–144*, 2004.

HEYMAN, Thomas et al. An Analysis of the Security Patterns Landscape. In: *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*. 2007.

JAFARI, Abbas Javan; RASOOLZADEGAN, Abbas. *Security Patterns: A Systematic Mapping Study*. 2018.

KIENZLE, Darrell M. et al. *Security Patterns Repository Version 1.0*. 2001.

LAFORE, Robert. *Object-Oriented Programming in C++ (4th Edition)*. Sams Publishing, 2001. ISBN: 0672323087.

MAHMOUD, Qusay H. Security Policy: A Design Pattern for Mobile Java Code. In: *Proceedings of the seventh conference on pattern languages of programming (PLoP '00)*. 2000.

MENDES, Antonio. *Introdução a Programação Orientada a Objetos com C++ (Em Portuguese do Brasil)*. Elsevier, 2010. ISBN: 978-8535237023.

MUIJNCK-HUGHES, Jan de; DUNCAN, Ishbel M. Issues Affecting Security Design Pattern Engineering. In: *Proceedings of Cyberpatterns*. 2013.

MUNAWAR HAFIZ, Ralph E. Johnson. *Security Patterns and their Classification Schemes*. 2006.

NETLAND, Lars-Helge; ESPELID, Yngve; MUGHAL, Khalid Azim. Security Pattern for Input Validation. In: *Proceedings of the Fifth Nordic Conference on Pattern Languages of Programs*. 2006.

PONDE, Poonam S.; SHIRWAIKAR, Shailaja C.; KHARAT, Vilas S. Knowledge representation of security design pattern landscape using formal concept analysis. *Journal of Engineering Science and Technology*, 2017.

PONDE, Poonam; SHIRWAIKAR, Shailaja; GORE, Sharad. Hierarchical Cluster Analysis On Security Design Patterns. In: *Proceedings of the International Conference on Advances in Information Communication Technology & Computing - AICTC '16*. ACM Press, 2016. DOI: <10.1145/2979779.2979871>.

PONDE, Poonam; SHIRWAIKAR, Shailaja; KREINER, Christian. An Analytical Study of Security Patterns. In: *Proceedings of the 21st European Conference on Pattern Languages of Programs*. 2016.

RICHARD WIENER, Lewis J. Pinson. *Fundamentals of OOP and Data Structures in Java*. Cambridge University Press, 2000. ISBN: 9780521662208,0-521-66220-6. Disponível em: <<http://gen.lib.rus.ec/book/index.php?md5=687AE66CBA6B39F181EDCC53C28981E7>>.

RIMBA, Paul et al. Composing Patterns to Construct Secure Systems. In: *2015 11th European Dependable Computing Conference (EDCC)*. IEEE, set. 2015. DOI: <10.1109/edcc.2015.12>.

ROMANOSKY, Sasha. *Security Design Patterns Part 1 v1.4*. 2001.

ROSADO, David G.; FERNÁNDEZ-MEDINA, Eduardo; PIATTINI, Mario. Comparison of Security Patterns. *IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.2B*, 2006.

RÖSER, Florian. *Security Design Patterns in Software Engineering - Overview and Example*. 2012. Tese (Mestrado) – University of Media, Stuttgart.

SCHUMACHER, Markus et al. *Security Patterns: Integrating Security and Systems Engineering*. Edição de John Wiley. John Wiley & Sons, Inc., 2005.

STAMP, Mark. *Information Security : Principles and Practice*. Wiley-Interscience, 2005. ISBN: 9780471738480.

STEEL, Chris; NAGAPPAN, Ramesh; LAI, Ray. *Security Patterns for J2EE Applications, Web Services, Identity Management and Service Provisioning*. 2003.

WEIDMAN, Georgia. *Penetration Testing*. No Starch Press, 2014. ISBN: 978-1-59327-564-8.

YODER, Joseph; BARCALOW, Jeffrey. Architectural Patterns for Enabling Application Security. In: *PLoPD-4 Book*. 1997.

_____-

APÊNDICE A – ARTIGO X REFERÊNCIA

Artigo	Referência
(HALKIDIS; CHATZIGEORGIOU; STEPHANIDES, 2004)	1
(HEYMAN et al., 2007)	2
(P. PONDE; S. SHIRWAIKAR; KREINER, 2016)	3
(YODER; BARCALOW, 1997)	4
(STEEL; NAGAPPAN; LAI, 2003)	5
(DANGLER, 2013)	6
(ROSADO; FERNÁNDEZ-MEDINA; PIATTINI, 2006)	7
(RIMBA et al., 2015)	8
(MUIJNCK-HUGHES; DUNCAN, 2013)	9
(P. PONDE; S. SHIRWAIKAR; GORE, 2016)	10
(P. S. PONDE; S. C. SHIRWAIKAR; KHARAT, 2017)	11
(BUNKE; KOSCHKE; SOHR, 2012)	12
(HAFIZ; ADAMCZYK; JOHNSON, 2007)	13
(DOUGHERTY et al., 2009)	14
(BLAKLEY; HEATH, 2004)	15
(ROMANOSKY, 2001)	16
(NETLAND; ESPELID; MUGHAL, 2006)	17
(JAFARI; RASOOLZADEGAN, 2018)	18
(RÖSER, 2012)	19
(MAHMOUD, 2000)	20
(MUNAWAR HAFIZ, 2006)	21
(ELDER, 2001)	22
(KIENZLE et al., 2001)	23

APÊNDICE B – PADRÕES LEVANTADOS - SIMPLIFICADA

Name	Mentions
A Pattern For Ws-trust	3; 10; 11; 12;
Access Control List (Acl)	3; 10; 11; 12;
Access Control Requirements	3; 10; 12; 19;
Access Controller	3; 10; 12;
Access Session	3; 10; 11; 12;
Account Lockout	3; 10; 12; 22; 23;
Actor And Role Lifecycle	3; 10; 12;
Address Book	3; 10; 12;
Administrator Hierarchy	3; 10; 12;
Administrator Objects	3; 10; 12;
Agency Guard	3; 10; 12;
Agent Authenticator	3; 10; 12;
Alice And Friends	3; 10; 12;
Anonymity Set	3; 10;
Application Firewall,	3; 10;
Assertion Builder	3; 5; 10; 12;
Asset Valuation	2; 3; 10;
Audit Interceptor	3; 5; 10; 12;
Audit Requirements,	3; 10; 19;
Audit Trail and Logging Requirements	19;
Authenticated Session,	3; 10; 12; 22; 23;
Authentication Enforcer	3; 5; 10; 12;
Authenticator	1; 3; 6; 10; 12; 13; 15; 19; 21;
Authoritative Source Of Data	3; 10; 16;
Authorization Enforcer	3; 5; 10; 12;
Authorization Pattern	12;
Authorization,	3; 6; 7; 10; 12; 13; 19; 21;
Automated Identification and Authentication Design Alternatives	3; 10; 19;
Batched Routing	3; 10;
Biometrics Design Alternatives	3; 10;
Brokered Authentication	3; 10;
Build The Server From The Ground Up	3; 10; 12; 22; 23;
Capability	3; 10; 11; 12;
Chaining,	3; 10;

Name	Mentions
Check Point,	3; 4; 6; 7; 10; 11; 12; 16; 19;
Checkpointed System	1; 3; 10; 12; 13; 15; 21;
Choose The Right Stuff	3; 10; 12; 22; 23;
Chrootjail	3; 10;
Clear Sensitive Information	3; 6; 10; 12; 14;
Client Data Storage	3; 10; 12; 22; 23;
Client Input Filters	3; 10; 12; 17; 22; 23;
Comparator-checked Fault Tolerant System	1; 3; 10; 15;
Compartmentalization,	3; 10; 12; 21;
Container Managed Security	3; 5; 10; 12;
Content Dependent Processing,	3; 10;
Content Independent Processing [83]	12;
Controlled Execution Environment	3; 10; 12; 19;
Controlled Object Factory	3; 6; 10; 12; 19;
Controlled Object Monitor	3; 10; 12; 19;
Controlled Process Creator	3; 10; 12; 19;
Controlled Virtual Address Space,	3; 10; 12; 19;
Cover Traffic	3; 10;
Credential Delegation	3; 10; 12;
Credential Tokenizer	3; 5; 10; 12;
Credential,	3; 10; 12;
Data Privacy, Integrity, Authentication	16;
Data Sanitization	16;
Defense In Depth	3; 10; 13; 15; 16; 21;
Defer To Kernel	3; 6; 10; 12; 14; 19;
Delayed Routing	3; 10;
Demilitarized Zone	12; 19;
Directed Session	3; 10; 12; 22; 23;
Distributed Responsibility	3; 10; 12;
Distributed Trust	16;
Distrustful Decomposition	3; 6; 10; 12; 14; 19;
Dmz,	3; 10;
Document The Security Goals	3; 10; 12; 22; 23;
Document The Server Configuration	3; 10; 12; 22; 23;
Dos Security	3; 10;
Dynamic Service Management	3; 5; 10; 12;
Encrypted Storage	3; 10; 12; 22; 23;

Name	Mentions
Enroll By Validating Out Of Band	3; 10; 12; 22; 23;
Enroll Using Third Party Validation	3; 10; 12; 22; 23;
Enroll With A Pre-existing Shared Secret	3; 10; 12; 22; 23;
Enroll Without Validating	3; 10; 12; 22; 23;
Enterprise Partner Communication	3; 10;
Enterprise Security Approaches	3; 10;
Enterprise Security Services	3; 10;
Error Detection/correction	1; 3; 10; 15;
Exception Manager	19;
Exception Shielding	3; 10; 13;
Execution Domain	3; 10; 12; 19;
Face To Face	3; 10; 12;
Fail Securely	3; 10; 16;
Fault Container	3; 10; 12;
Feel the Network	16;
File Authorization	3; 7; 10; 12; 19;
Filter Firewall,	3;
Firewall	3; 10;
Front Door	3; 10; 12;
Full Access With Errors	3; 10; 11; 12; 19; 21;
Full View With Errors	4; 6; 12; 16;
Grant-based Access Control Pattern (Gbac)	12;
Grey Hats	13;
Hidden Implementation	3; 10; 12; 22; 23;
Hidden Metadata	3; 10;
Ia Design Alternatives	3; 10;
Ia Requirements	3; 10;
Id Password Authentication	3; 10; 12;
Identification and Authentication Requirements	19;
Information Encryption	19;
Information Obscurity	3; 6; 10; 12;
Input Guard	3; 10; 12;
Input Validation	3; 6; 10; 12; 14; 17; 19;
Integration Reverse Proxy	3; 10; 19;
Intercepting Validator	3; 5; 10; 12; 17;
Intercepting Web Agent	3; 5; 10; 12;
Intrusion Detection Requirements	3; 10; 19;

Name	Mentions
Keep Session Data In Client	3; 10; 12;
Keep Session Data In Server	3; 10; 12;
Key In The Pocket	3; 10; 12;
Known Partners	3; 10; 12; 19;
Layered Encryption,	3; 10;
Layered Security	16;
Least privileges	16;
Limited Access	12; 19;
Limited Access With Errors	3; 10; 11;
Limited View	4; 6; 12; 16;
Link Padding,	3; 10;
Load Balancer	3; 10; 12;
Log For Audit	3; 10; 12; 22; 23;
Low Hanging Fruit	3; 10; 16;
Message Inspector	3; 5; 10; 12;
Message Interceptor Gateway,	3; 5; 10; 12;
Message Replay Detection	3;
Metadata-Based Access Control (MBAC)	19;
Minefield,	3; 10; 12; 13; 21; 22; 23;
Morphed Representation	3; 10;
Multilevel Secure Partitions	3; 10; 11; 12;
Multilevel Security	3; 6; 7; 10; 11; 12; 19;
Multiple Secure Observers Using J2ee	12;
Network Address Blacklist	3; 10; 12; 22; 23;
Non Repudiation Requirements	3; 10; 19;
Obfuscated Transfer Object	3; 5; 10; 12;
Oblivious Transfer	3; 10;
Output Guard	3; 10; 12;
Output Validation	19;
Packet Filter Firewall	3; 10;
Partitioned Application	3; 10; 12; 22; 23;
Password Authentication,	3; 10; 11; 12; 22; 23;
Password Design And Use	3; 10; 19;
Password Propagation	3; 10; 12; 22; 23;
Password Synchronizer	3; 5; 10; 12; 13;
Patch Proactively	3; 10; 12; 22; 23;
Pathname Canonicalization	3; 6; 10; 12; 14;

Name	Mentions
Policy	1; 3; 10; 12; 15;
Policy Based Access Control	3; 10; 12;
Policy Delegate	3; 5; 10; 12;
Policy Enforcement	3; 10; 12;
Policy Enforcement Point	13; 21;
Privilege Limited Role	3; 10; 12;
Privilege Separation(PrivSep)	3; 6; 10; 12; 14; 19;
Protected Entry Points	3; 10; 12;
Protected System	1; 3; 10; 12; 15;
Protection Reverse Proxy	3; 10; 19;
Protection Rings	3; 10; 12;
Proxy Based Firewall	3; 10;
Red Team The Design	3; 10; 12; 22; 23;
Reference Monitor	3; 7; 10; 12; 19;
Replicated System	1; 2; 3; 10; 13; 15; 21;
Resource Acquisition Is Initialization (Raii)	3; 6; 10; 12; 14;
Risk Assessment and Management	16;
Risk Determination	3; 10;
Role Based Access	2; 12;
Role Based Access Control	3; 6; 7; 10; 12; 19;
Role Hierarchies	3; 10; 12;
Role Rights Definition	3; 6; 10;
Role Validator	3; 10; 12;
Roles	2; 3; 4; 6; 10; 12; 16;
Rule Set Based Access Control (RSBAC)	19;
Safe Data Buffer	13;
Safe Data Structure	3; 10;
Sandbox	3; 10; 12; 19;
Seal Ring Engraver	3; 10; 12;
Sealed Envelope	3; 10; 12;
Sealed Signed Envelope	3; 10; 12;
Secure Access Layer	3; 4; 6; 10; 12; 16; 19;
Secure Accounting Requirements	10;
Secure Assertion	3; 10; 12; 22; 23;
Secure Base Action	3; 5; 10; 12;
Secure Blackboard	19;
Secure Broker	3; 12;

Name	Mentions
Secure Builder Factory	3; 6; 10; 12; 14; 19;
Secure Chain Of Responsibility,	3; 6; 10; 12; 14; 19;
Secure Channels	3; 6; 10; 12; 19;
Secure Communication	1; 2; 3; 10; 12; 15;
Secure Directory,	3; 6; 10; 12; 14;
Secure Factory	3; 6; 10; 12; 14;
Secure Logger	3; 5; 6; 10; 11; 12; 14; 19;
Secure Message Router	3; 5; 10; 12;
Secure Pipe	3; 5; 10; 12;
Secure Preforking	3; 10; 12; 13; 21;
Secure Process Thread	3; 10; 11; 12;
Secure Proxy	1; 15;
Secure Resource Pooling	3; 10;
Secure Service Facade	3; 5; 10; 12;
Secure Service Proxy,	3; 5; 10; 12;
Secure Session Manager,	3; 5; 10;
Secure Session Object	3; 5; 10; 12;
Secure State Machine,	3; 6; 10; 12; 14; 19;
Secure Strategy Factory	3; 6; 10; 12; 14;
Secure Visitor	3; 6; 10; 11; 12;14; 19;
Security Accounting Requirements	3; 10; 19;
Security Association	1; 3; 10; 12; 15;
Security Context	1; 3; 10; 12; 15;
Security Needs Identification	3; 10;
Security Policy: A Design Pattern For Mobile Java Code	12; 20;
Security Session	3; 10; 12; 19;
Server Sandbox	3; 10; 12; 22; 23;
Session Based Attribute Based Authorization	3; 10; 12;
Session Based Role Based Access Control	3; 10; 12; 16;
Session Failover	3; 10; 12;
Session Management	3; 10; 12;
Session Scope	3; 10; 12;
Session Timeout	3; 10; 12;
Session/secure Session[6]	3; 4; 6; 7; 10; 12; 16;
Share Responsibility For Security	3; 10; 12; 22; 23;
Signed Envelope	3; 10; 12;
Single Access Point	3; 4; 6; 7; 10; 12; 13; 16; 19; 21;

Name	**Mentions**
Single Session	3; 10; 12;
Single Sign On	3; 10;
Single Sign On Delegator	3; 5; 10; 12;
Single Threaded Facade	21;
Small Processes	3; 10;
Standby	1; 2; 15;
Stateful Firewall	3; 10;
Subject Description	12;
Subject Descriptor	1; 3; 10; 12; 13; 15; 21;
Symmetric Encryption	12;
Tandem System	3; 10; 15;
Test On A Staging Server	3; 10; 12; 22; 23;
The Forged Seal Ring	3; 10; 12;
The Real Thing	3; 10; 12;
The Security Provider	16;
There Is Somebody Eavesdropping	3; 10; 12;
Third Party Communication	3; 10; 16;
Threat Assessment Pattern	3; 10;
Trust Partitioning	3; 10; 21;
Trusted Proxy	3; 10; 12; 22; 23;
Unique Entry Of Information	3; 10; 12;
Validated Transactions	3; 10; 12; 22; 23;
Virtual Address Space Access Control	3; 7; 10; 12;
Virtual Address Space Structure Selection	3; 10; 12;
Virtualization	19;
Vulnerability Assessment	3; 10;
White Hats Hack Thyself	3; 10; 16;
Ws Trust	3;
Xml Encryption	3; 10; 11; 12;

Tabela 3 – Tabela dos Padrões levantados.

APÊNDICE C – FILTRAGEM 1

Tabela 4 – Classificação Final dos Padrões

Name	Mentions	Peso_Mentions	#_Mentions	Ment_Quali	Ment_Quant	#_Diagr	Peso_Total
Single Access Point	3; 4; 6; 7; 10; 12; 13; 16; 19; 21;	11.25	10	5	5	2	13.25
Check Point,	3; 4; 6; 7; 10; 11; 12; 16; 19;	11	9	5	4	3	14
Authenticator	1; 3; 6; 10; 12; 13; 15; 19; 21;	9.25	9	4	5	3	12.25
Input Validation	3; 6; 10; 12; 14; 17; 19;	8.75	7	4	3	5	13.75
Secure Access Layer	3; 4; 6; 10; 12; 16; 19;	8.75	7	4	3	2	10.75
Session/secure Session	3; 4; 6; 7; 10; 12; 16;	8.75	7	4	3	1	9.75
Authorization,	3; 6; 7; 10; 12; 13; 19; 21;	7.25	8	3	5	2	9.25
Secure Logger	3; 5; 6; 10; 11; 12; 14; 19;	7.25	8	3	5	3	10.25
Multilevel Security	3; 6; 7; 10; 11; 12; 19;	7	7	3	4	2	9
Roles	2; 3; 4; 6; 10; 12; 16;	7	7	3	4	1	8
Secure Visitor	3; 6; 10; 11; 12;14; 19;	7	7	3	4	4	11
Checkpointed System	1; 3; 10; 12; 13; 15; 21;	5.25	7	2	5	3	8.25
Replicated System	1; 2; 3; 10; 13; 15; 21;	5.25	7	2	5	3	8.25
Subject Descriptor	1; 3; 10; 12; 13; 15; 21;	5.25	7	2	5	3	8.25
Minefield	3; 10; 12; 13; 21; 22; 23;	3.5	7	1	6	0	3.5

APÊNDICE D – FILTRAGEM 2

Pattern	# Mentions	Availability	Confidentiality	Integrity
A Pattern For Ws Trust	1		x	
Access Control List (Acl)	1		x	
Access Controller	1		x	x
Access Session	1		x	x
Actor And Role Lifecycle	1		x	
Address Book	1			x
Administrator Hierarchy	1		x	
Administrator Objects	1			x
Agency Guard	1		x	x
Authenticator**	3		x	
Authorization**	3		x	
Build The Server From The Ground Up	1	x		x
Capability	1		x	x
Check Point**	2		x	x
Checkpointed System**	3	x		x
Choose The Right Stuff	1	x		x
Clear Sensitive Information	2		x	
Client Data Storage	1		x	x
Client Input Filters	1			x
Compartmentalization	2	x	x	x
Content Independent Processing	1		x	x
Controlled Execution Environment	1			x
Controlled Object Factory	1			x
Controlled Object Monitor	1			x
Controlled Process Creator	1		x	x
Controlled Virtual Address Space,	1			x
Credential Delegation	1		x	
Defense In Depth	2	x	x	x
Defer to Kernel	2		x	x
Demilitarized Zone	1		x	
Directed Session	1			x
Distributed Responsibility	1		x	x
Distrustful Decomposition [	2			x
Document The Server Configuration	1	x		x

Pattern	# Mentions	Availability	Confidentiality	Integrity
Encrypted Storage	1		x	x
Enroll By Validating Out Of Band	1	x	x	x
Enroll Using Third Party Validation	1	x	x	x
Enroll With A Pre existing Shared Secret	1	x	x	
Exception Shielding	1		x	
Execution Domain	1			x
Fault Container	1			x
File Authorization	1		x	x
Front Door	1		x	
Full Access With Errors	2		x	x
Full View with Errors	2	x	x	x
Grey Hats	1	x	x	x
Hidden Implementation	1	x	x	
Information Obscurity	2		x	
Input Guard	1			x
Input Validation**	2			x
Intercepting Validator	1			x
Keep Session Data In Client	1	x		
Key In The Pocket	1			x
Limited Access	1		x	x
Limited View [	2	x	x	
Load Balancer	1	x		
Log For Audit	1			x
Message Interceptor Gateway,	1		x	
Minefield**	3	x	x	x
Multilevel Secure Partitions	1		x	x
Multilevel Security**	2		x	x
Multiple Secure Observers Using J2EE	1		x	
Network Address Blacklist	1		x	x
Obfuscated Transfer Object	1			x
Output Guard	1			x
Partitioned Application	1	x	x	
Password Authentication,	1		x	x
Password Propagation	1		x	x
Password Synchronizer	1	x		x

Pattern	# Mentions	Availability	Confidentiality	Integrity
Pathname Canonicalization	2			x
Policy	1		x	
Policy Delegate	1		x	
Policy Enforcement	1		x	x
Policy Enforcement Point	2		x	
Privilege Limited Role	1			x
Privilege Separation	1			x
Protection Rings	1			x
Reference Monitor	1		x	x
Replicated System**	2	x		x
Resource Acquisition is Initialization (RAII)	2	x	x	x
Role Based Access	1		x	x
Role Hierarchies	1		x	
Role Rights Definition	1		x	
Role Validator	1		x	
Role Based Access Control	1		x	
Roles**	1		x	
Safe Data Buffer	1			x
Sandbox	1			x
Sealed Envelope	1		x	x
Sealed Signed Envelope	1		x	x
Secure Access layer**	2			x
Secure Assertion	1			x
Secure Assertion	1	x		
Secure Builder Factory	2			x
Secure Chain of Responsibility	2		x	x
Secure Channels	2		x	x
Secure Communication	1		x	
Secure Directory	2			x
Secure Factory	2		x	x
Secure Message Router	1			x
Secure Pipe	1			x
Secure Preforking	3	x	x	x
Secure Process Thread	1		x	x
Secure Service Facade	1		x	
Secure State Machine	2	x	x	x

Pattern	**# Mentions**	**Availability**	**Confidentiality**	**Integrity**
Secure Strategy Factory	2		x	x
Secure Visitor**	2		x	x
Security Association	1			x
Security Context	1		x	
Security Policy: A Design Pattern For Mobile Java Code	1			x
Security Session	1			x
Server Sandbox	1			x
Session Based Role Based Access Control	1			x
Session Failover	1	x		
Session Management	1		x	x
Session Scope	1	x	x	
Session Timeout	1	x		x
Session/Secure Session**	2		x	x
Signed Envelope	1		x	x
Single Access Point**	3		x	
Single Session	1			x
Single Sign On Delegator	1		x	
Single Threaded Facade	1		x	
Subject Descriptor**	3		x	
Symmetric Encryption	1		x	x
Test On A Staging Server	1	x		
The Forged Seal Ring	1			x
Trust Partitioning	1		x	
Trusted Proxy	1			x
Validated Transactions	1			x
Virtual Address Space Access Control	1		x	x
Virtual Address Space Structure Selection	1			x
Xml Encryption	1		x	x

Tabela 5 – Tabela dos Padrões por Pilar