UNIVERSIDADE FUMEC
FACULDADE DE CIÊNCIAS EMPRESARIAIS - FACE

THIAGO MARQUES BARCELOS

SEGURANÇA EM APLICAÇÕES WEB:

Exploração de vulnerabilidades em aplicações utilizando técnicas

BELO HORIZONTE
2010

THIAGO MARQUES BARCELOS

SEGURANÇA EM APLICAÇÕES WEB:

Exploração de vulnerabilidades em aplicações utilizando técnicas

Monografia apresentada à Universidade FUMEC, no curso de Ciência da Computação, apresentado à disciplina Trabalho de Conclusão de Curso.

Orientadores:
Professor Osvaldo Manoel Corrêa
Professor Ricardo Terra Nunes Bueno Villela

BELO HORIZONTE
2010

RESUMO

A evolução mundial trouxe consigo a necessidade de informatizar dezenas de tarefas. Essa informatização, em sua grande maioria, se deu através do desenvolvimento de aplicações *web*. Essas aplicações são responsáveis pelo gerenciamento de inúmeras informações e serviços vitais a sociedade, como sistemas bancários, sistemas de saúde, sistemas de universidades, sistemas de órgãos governamentais etc. Tendo em vista a importância dessas ferramentas, pode-se imaginar o impacto que uma interrupção em seus serviços poderia causar.

É nesse contexto que a segurança se enquadra. Sem ela, não há como garantir a integridade, confidencialidade e disponibilidade dos serviços e dados disponíveis. Mesmo diante dessa preocupação ainda existem aplicações inseguras que são exploradas através de técnicas conhecidas e muito poderosas.

Diante disso, existem certos cuidados que podem e devem ser tomados durante o desenvolvimento de sistemas de software que ajudam a protegê-las contra esses ataques. Assim, ao se pensar em disponibilizar serviços e informações na rede mundial - a Internet, a segurança deve merecer um destaque especial.

Palavras chave: Segurança. Aplicações. Ataques. Exploração. Vulnerabilidades.

ABSTRACT

Brough along the need to automate dozens of tasks. This computerization mostly occurred through the development of *web* applications. These applications are responsible for managing the wealth of information and vital services to society as, banking systems, health systems, university systems, government systems etc.

Given the importance of these tools, imagined the impact that an interruption in the services provided by them could cause.

In this context, security comes into play. Without it there is no way to guarantee the integrity, confidentiality and reliability of services and data. Even before this there is still concern that unsafe applications are explored through techniques known and very powerful.

There are precautions that can be taken during the development of software systems that help protect them against these attacks. When you think of the services and information in the Internet (World Wide *Web*), safety deservers special attention.

Keywords: Security. Applications. Attacks. Exploration. Vulnerabilities.

LISTA DE FIGURAS

LISTA DE SIGLAS

API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
CRLF	C*arriage Return Line Feed*
CPF	*Cadastro de Pessoa Física*
CNPJ	*Cadastro de Pessoa Jurídica*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IP	*Internet Protocol*
ISP	*Internet Services Provider*
MAC	*Media Access Control*
MVC	*Model-View-Controller*
RFC	*Request for comments*
SQL	*Structured Query Language*
SSL	*Secure Socket Layer*
TCP	*Transmission Control Protocol*
URL	*Uniform Resource Locator*

SUMÁRIO

INTRODUÇÃO

Com o grande volume de informações, sistemas e serviços disponíveis na Internet, a taxa de crescimento de falhas de segurança é grande, principalmente nos sistemas *web*. Essas falhas e vulnerabilidades permitem a exploração com fins maliciosos que podem trazer grandes consequências.

Diante da evolução tecnológica e dos sistemas criados para sua gerência, muitos ainda desconhecem a importância da segurança e quais são os impactos que um leve descuido pode trazer. Como fator primordial, a segurança deve ser levada em consideração para a elaboração de qualquer ferramenta *web* que disponibilize informações, serviços e principalmente se for foco do negócio da entidade patrocinadora. Um bom nível de segurança é obtido quando três princípios básicos são respeitados: Confidencialidade, Integridade, Disponibilidade.

Os sistemas *web* estão presentes em grande parte das atividades econômicas da sociedade e com a ajuda desses sistemas é que a disponibilização de informações e serviços pode ser realizada. Sem esses sistemas, a sociedade atual não conseguiria que o mercado tivesse tamanha abrangência e que a circulação de informações fosse tão rápida. Assim, cresce a dependência e a publicação de informações na Internet, sendo muito importante a adoção de meios de segurança para que esses sistemas tornem-se confiáveis. Considerando nesse contexto este trabalho apresenta e explica as vulnerabilidades existentes nos sistemas atuais, apresenta as técnicas de exploração mais utilizadas e algumas ferramentas e métodos que garantem um nível maior na segurança das aplicações *web*.

Para entender como funcionam as aplicações *web* é necessário conhecer alguns conceitos básicos, além realizar um estudo do funcionamento das aplicações *web*. Este trabalho é constituído de quatro capítulos organizados da seguinte maneira. No capítulo 1, Segurança em aplicações *web*, são introduzidos conceitos básicos sobre a Internet, protocolos de comunicação, funcionamento de aplicações *web*, além apresentar fundamentos básicos de segurança. No capítulo 2, Técnicas de exploração, é apresentado o funcionamento das principais técnicas de exploração de vulnerabilidades em aplicações *web.* No capítulo 3, Aplicabilidade, é apresentada a aplicação de cada técnica de exploração, seguido dos estudos de caso. No capítulo 4, Formas de proteção, são apresentados ferramentas, métodos e práticas de programação que ajudam na prevenção de ataques de exploração.

Através deste trabalho é possível identificar os problemas mais frequentes que ocorrem em aplicações *web,* entender como ocorrem, analisar a aplicação prática em casos reais e aplicar as possíveis soluções para evitar que falhas ocorram. Além de ressaltar a importância do papel que a segurança desempenha nas aplicações e como realizar testes para garantir que a sua aplicação está segura.

Capítulo 1 – Segurança em aplicações web

Com o crescente uso das ferramentas providas pela informática, encontramos inúmeros serviços e informações disponíveis na grande rede mundial: a Internet. Esses recursos podem ser acessados em qualquer localização geográfica através de dispositivos portáteis, *notebooks*, computadores, televisões e outros equipamentos. (KUROSE, 2006).

Conforme Mendes (2007), esse grande uso é justificado pela facilidade, comodidade, praticidade e abrangência disponibilizada por esses recursos. Essa demanda trouxe a necessidade de garantir a segurança dos serviços e informações disponibilizados na Internet, garantindo a confidencialidade, integridade e disponibilidade dos serviços, mesmo em situações extremas. (LEHTTINEN; RUSSELL; GANGEMI, 2006).

Todos os serviços providos sem segurança pela Internet estão sujeitos a falhas, invasões, alterações e, até mesmo, a exclusão. Esse cenário torna-se preocupante quando tratamos de aplicações *web*, que geralmente são utilizadas para armazenar, disponibilizar informações e serviços, contendo dados sigilosos e importantes. Mesmo com ferramentas disponíveis para o desenvolvimento de aplicações *web,* existem várias falhas de segurança que, se exploradas, podem permitir a entrada não autorizada comprometendo o seu funcionamento e a confiabilidade de seus serviços.

Este capítulo está organizado conforme a seguir. Na seção 1.1, é abordada a história da Internet, suas características, os protocolos mais utilizados, uma síntese da arquitetura TCP/IP (*Transmission Control Protocol*) e o que significa o endereço de páginas na Internet: a URL (*Uniform Resource Locator*). Na seção 1.2, é conceituado o funcionamento de aplicações *web*, explicado a atuação do protocolo HTTP (*Hypertext Transfer Protocol*), funcionamento da arquitetura cliente-servidor e uma breve descrição das linguagens mais comuns para programação em ambiente *web*. Na seção 1.3, é fundamentado o conceito de segurança, além de abordar sobre o protocolo utilizado para prover proteção no ambiente *web*: o SSL (*Secure Socket Layer*). Para a proteção de informações, essa seção explica de forma sucinta uma técnica muito utilizada: a criptografia.

1.1 Internet

Como consequência de um experimento ao término da década de 1960, a Agência de Projetos de Pesquisas Avançadas (ARPA) do Departamento de Defesa do Estados Unidos da América (DARPA) interligou computadores em uma rede de longa distância denominada ARPANET (*Advanced Research Projects Agency Network*) espalhando-se por todo o território americano.

De acordo com Mendes (2007), o objetivo da ARPANET era permitir que as forças armadas e governamentais trocassem informações, compartilhassem arquivos, programas e principalmente recursos de hardware. Para aprimorar e padronizar a forma de comunicação dentro da ARPANET, na década de 1980, foi desenvolvida a arquitetura TCP/IP. Com o desenvolvimento dessa arquitetura-padrão, vários sistemas operacionais que executavam em vários computadores puderam se adequar e assim se conectar a ARPANET. Com esse avanço, a rede experimental cresceu exponencialmente sendo conhecida como uma confederação de redes locais, chamada Internet. Ainda segundo Mendes (2007), após anunciar ao fim do experimento pela DARPA, a Fundação Nacional de Ciência (*National Science Foundation*) criou outra rede denominada NSFNET substituindo ARPANET, a conhecida Internet.

1.1.1 Características

De acordo com Kurose (2006), a Internet possibilita a interconexão de milhões de equipamentos de computação em todo o mundo: computadores, televisões, automóveis, celulares, agendas digitais, sistemas elétricos, entre outros. No dialeto da Internet, todos esses equipamentos são denominados hospedeiros ou sistemas finais.

Sistemas finais são interligados por enlaces de comunicação permitindo assim o tráfego de dados. Cada sistema final acessa a Internet através de um Provedor de Serviços de Internet (*Internet Service Provider* – ISP), que são interligados convergindo todos em único ambiente. Dentro desse ambiente, encontramos sistemas *web* que permitem a troca de dados entre os sistemas finais, permitindo que exista a troca instantânea de mensagens, vídeos, áudio, além de disponibilizar serviços de telefonia, entre outros.

1.1.2 Protocolos

Conforme Kurose (2006), o principal papel de um protocolo é padronizar como devidos serviços irão funcionar. Esse processo de padronização é realizado pela *Internet Engineering Task Force* (Força de Trabalho de Engenharia de Internet, IETF). Todo esse feito gera documentos denominados *request for comments* (pedido de comentários, RFCs) abrangendo todo o funcionamento detalhado do protocolo.

Ainda segundo Kurose (2006), diante da importância que possuem os protocolos, é importante apresentar alguns dos mais utilizados na Internet, para a utilização *web* foram criados o TCP (*Transmission Control Protocol*), IP (*Internet Protocol*), HTTP (*Hypertext Transfer Protocol*) e o SMTP (Protocolo Simples de Transferência – *Simple Mail Transfer Protocol*). Todos eles são importantes, mas graças ao uso coletivo do TCP/IP, é que a Internet pode disponibilizar inúmeros recursos.

1.1.3 Arquitetura TCP/IP

Para possibilitar o avanço da Internet foi necessário o desenvolvimento do TCP/IP. O TCP/IP é o conjunto de dois protocolos não independentes. Segundo Scrimger, "um conjunto de protocolos é uma coleção hierárquica de protocolos relacionados. Por causa do papel revolucionário que o TCP e o IP desempenharam no avanço da rede, o conjunto inteiro é referido como conjunto de protocolo TCP/IP." (SCRIMGER, 2002, p.31).

Segundo Scrimger (2002), a arquitetura TCP/IP é constituída por um conjunto de camadas que pode ser observado na FIG. 1, onde cada uma é responsável por tarefas específicas.

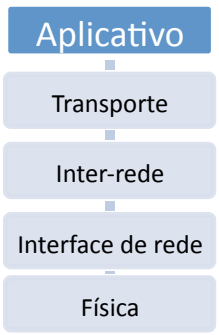

FIGURA 1 - Organização das camadas da arquitetura TCP/IP.

Fonte: Do autor.

As camadas que consistem o TCP/IP são:

1. Camada Física: considerada a camada mais baixa do modelo, é responsável pela transmissão física dos dados. Essa tarefa é realizada com conversão dos dados em uma série de *bits*. Após esse trabalho, os *bits* são convertidos em pulsos elétricos e enviados através do enlace de transmissão, como cabos elétricos, cabos de fibra óptica, ondas de rádios etc.
2. Interface de rede: a sua principal função é a identificação dos dispositivos em uma rede local através do endereço MAC.
3. Inter-rede: utilizando o protocolo IP, realiza o roteamento dos pacotes de dados possibilitando a comunicação entre os dispositivos da rede local.
4. Transporte: fornece uma interface de comunicação entre as camadas inferiores (Física, Interface de rede e Inter-rede) com a camada de Aplicativo.
5. Aplicativo: é a camada que interage diretamente com o usuário e suporta todos os protocolos requeridos para prover os serviços de rede.

1.1.4 Uniform Resource Locator (URL)

A URL é considerada um endereço global. É utilizada para que seja possível encontrar qualquer tipo de informação disponível na Internet, como páginas *HyperText Markup Language* (HTML), áudio, vídeo, imagens etc. Segue o exemplo de uma URL:

```
http://www.exemplo.com.br/index.html
```

Conforme Freeman (2005), a URL geralmente é composta de três partes: a primeira parte identifica o tipo de protocolo que será utilizado na solicitação de recursos, nesse exemplo corresponde a `http`; a segunda parte é o nome do *site* desejado `www.exemplo.com.br`; e, por fim, o caminho absoluto do recurso desejado `/index.html`.

1.2 Aplicações *web*

Com o grande número de usuários existentes na Internet, a cada momento são desenvolvidas aplicações que disponibilizam serviços e a troca de informações.

Essas aplicações são a evolução dos conhecidos aplicativos de computadores de mesa (*desktop*) que eram instalados, configurados em cada uma das máquinas que os utilizava. Além do grande custo com a manutenção, era necessário possuir os dados de instalação do sistema. Como a tendência de toda evolução é trazer benefícios, podemos dizer que os sistemas *web* revolucionaram a forma de utilização de recursos providos pela Internet. Com o uso em larga escala dos sistemas *web,* hoje é impossível falar em Internet e não notar a importância que esses sistemas representam.

Atualmente, empresas disponibilizam grande parte de seus serviços na Internet. Provendo agilidade, comodidade e praticidade ao seu cliente. É fácil conseguirmos movimentar contas bancárias, trocar mensagens em tempo real - seja áudio ou vídeo - a partir de algum dispositivo que esteja conectado a rede mundial. As barreiras da distância já não existem, além de não existir a necessidade de instalações no ambiente do cliente, pois a instalação é feita somente nos servidores e disponibilizadas através de endereços *web*.

A partir desse cenário, observa-se a importância e responsabilidade desses sistemas, pois manipulam informações sigilosas, importantes e de alto valor comercial.

1.2.1 Funcionamento

Grande parte dos sistemas *web* existentes são baseados na arquitetura cliente-servidor, onde os clientes solicitam as informações do servidor, e ele os responde com as informações solicitadas.

Segundo Lee (2005), os sistemas *web* são desenvolvidos em camadas. O código geralmente é divido em módulos de acordo com sua funcionalidade: interface com usuários, forma de negócio e conexão a base de dados.

Ainda conforme Lee (2005), seguindo esse padrão arquitetural, a camada de código que interage com o usuário é denominada apresentação. Já a que trata a lógica do sistema é chamada de controle. E a última camada, denominada modelo, geralmente é responsável pela comunicação com o banco ou fonte de dados. Em sistemas de software bem

desenvolvidos e planejados geralmente não existem mais do que essas três camadas, até porque quanto maior for o número desses módulos maior será a dificuldade em gerenciá-los. Esse padrão de desenvolvimento é conhecido como MVC (*Model-View-Controller* ou Modelo-Visão-Controlador).

> O MVC considera três papéis. O modelo é um objeto que representa alguma informação sobre o domínio. É um objeto não-visual contendo todos os dados e comportamento que não são usados pela interface de usuário. [...] A visão representa a exibição do modelo na interface do usuário. [...] O controlador recebe a entrada do usuário, manipula o modelo e faz com que a visão seja atualizada apropriadamente. Dessa forma, a interface de usuário é uma combinação da visão e do controlador. (FOWLER, 2006, p.315).

Com a divisão do código em módulos, a reutilização fica fácil. Porém, somente com esse artefato não conseguimos ter uma arquitetura escalável. Para que isso se torne possível utiliza-se o método de distribuir o código entre diversas máquinas, essa divisão é conhecida como filas. Essa segmentação em filas consiste em alocar os módulos do código em máquinas diferentes em uma infraestrutura de servidores distribuídos e, essa façanha é facilitada se o código da aplicação já estiver separado em camadas.

Dessa forma as três camadas básicas são alocadas em filas de acordo com sua funcionalidade e essas filas podem possuir números variados de servidores, que garantirão a escalabilidade, desempenho e disponibilidade a aplicação. Para os clientes o acesso é feito normalmente. Eles não conhecem a organização estrutural da aplicação. Para acessar basta que exista a conexão com a rede mundial através de algum dispositivo: celular, notebook, computador, televisão etc.

As informações podem trafegar em ambas as direções entre um cliente e um servidor. Normamente, a requisição parte de um cliente e o servidor devolve a resposta. Essa organização e a utilização da arquitetura cliente servidor pode ser observada na FIG. 2.

FIGURA 2 - Organização da arquitetura cliente servidor e as camadas de rede.
Fonte: Do autor.

Contudo, existem casos onde há a necessidade de uma troca contínua de informações entre o cliente e o servidor. É nesse momento que ambos os lados trafegam dados. A maior parte dos programas aplicativos utiliza um protocolo de transporte para se comunicar.

De acordo com Comer (2007), é dessa maneira que as aplicações *web* funcionam. Através de requisições e repostas que trafegam pelas camadas da arquitetura TCP/IP, seguindo pelo enlace e garantindo a comunicação entre as partes cliente e servidor.

1.2.2 Protocolo HTTP

Para que haja a comunicação entre o cliente e o servidor, as aplicações utilizam o protocolo HTTP. O cliente envia mensagens HTTP e o servidor o responde com base no mesmo protocolo. Geralmente, o protocolo utiliza a porta TCP 80 para receber as solicitações. A partir do momento em que existe a comunicação, o cliente é considerado um cliente *web.*

Segundo Costa (2008), para que não haja perda de informações durante a comunicação o protocolo HTTP utiliza de serviços localizados na camada de transporte do protocolo TCP.

Ainda conforme Costa (2008), uma mensagem HTTP é constituída por um cabeçalho contendo a identificação dos serviços que serão utilizados para a continuidade da tarefa, além de algumas especificações de controle pelo corpo da mensagem. No corpo da mensagem é que ficam armazenadas as informações trocadas entre o cliente e o servidor. No cabeçalho de uma mensagem pode constar qualquer um dos serviços descritos a seguir:

1. CONNECT: Utilizado em mensagens quando se existe um gerenciador de conteúdo de acessos a páginas web chamado de servidor Proxy.
2. DELETE: Solicitação de exclusão de algum recurso mantido pelo servidor.
3. GET: Solicita ao servidor algum recurso que ele disponibiliza.
4. HEAD: Solicita que o servidor envie informações sobre o recurso.
5. OPTIONS: Fornece a indicação dos serviços HTTP que podem ser utilizados para solicitações ao servidor.
6. POST: Envia ao servidor informações contidas em páginas web.
7. PUT: Instrução que faz a modificação de algum recurso do servidor.

8. TRACE: Permite que o cliente saiba quais servidores estão intermediando sua comunicação com o servidor de destino e se houve alguma injeção de informações a mensagem.

Segue um exemplo de como é o cabeçalho de uma mensagem HTTP:

```
GET /index.html HTTP/1.1
Host:www.exemplo.com.br
User-Agent: Mozilla/3.0
Accept: text/html, image/gif, image/jpeg
```

No exemplo acima é solicitado um recurso utilizando o método GET através da instrução `GET /index.html` utilizando a versão `HTTP 1.1`. A instrução `Host:www.exemplo.com.br` tem como função identificar o destino da requisição. Já identificação de qual navegador você está utilizando é representada pela instrução `User-Agent: Mozilla/3.0`. O tipo de conteúdo solicitado é feito através da linha Accept: `text/html, image/gif, image/jpeg.`

Segundo Costa (2008), quando ocorre alguma falha na solicitação, é reportada uma mensagem HTTP contendo algum dos códigos, que identificam as possíveis falhas:

1. 100-199: Reportam erros de ações ou situações temporárias.
2. 200-299: Indicação de sucesso.
3. 300-399: Informa que ocorreu algum redirecionamento durante a transmissão.
4. 400-499: Indica que ocorreu algum erro do lado cliente da comunicação. Um desses códigos mais conhecidos é o erro 404 que ocorre quando uma página não é encontrada.
5. 500-599: Referência de erros ocorridos pelo lado do servidor.
6. 600-699: Indica erros ocasionados por falhas gerais.

1.2.3 A troca de informações entre cliente-servidor

Para que haja interação entre o cliente e o servidor, deve existir alguma requisição pelo lado cliente. Essa requisição pode ser realizada através de um navegador ao inserir a URL de um documento. Quando existe interação com o servidor *web,* ambos os lados começam a troca de informações utilizando o HTTP. Por trás dessa conexão, existe uma complexa estrutura entre ambos os lados para controlar o *status* e integridade das

informações. Os pedidos realizados ao servidor são enviados através de quatro tipos de operações básicas suportadas pelo HTTP: GET, HEAD, POST ou PUT.

Segundo Comer (2007), quando o usuário solicita um *link* ou uma URL, é gerada uma instrução GET que utiliza a própria URL para o envio dos dados para o servidor com o seguinte formato: `GET item version CRLF` onde o `GET` representa o tipo da requisição, item representa o item requerido, *`version`* representa a versão do HTTP (que pode ser 1.0 ou 1.1) e `CRLF` compreende os caracteres da tabela *American Standard Code for Information Interchange* (ASCII) *carriage return line feed.*

O método GET faz com que as informações sejam passadas na própria URL: `www.exemplo.com.br?nome=Thiago` dessa maneira a instrução passada é que o campo `nome` está recebendo o valor `Thiago`.

O método POST utiliza o corpo da mensagem como meio de tráfego das informações enviadas pela página. Por isso ele é mais seguro que o método GET e permite que informações longas sejam enviadas:

```
POST /sistema/cadastro/cliente/RequestExe HTTP/1.1
HOST:
Content-type: application/x-www-form-urlencoded
Content-length: 32

firstname=Thiago&lastname=Barcelos
```

De acordo com a RFC2616[1], o método HEAD é muito parecido com o GET, ao utilizá-lo o servidor não responderá a requisição incluindo o corpo da mensagem, ele só retornará o cabeçalho da mensagem. A semelhança com o método GET também ocorre com o *PUT*, porém ele tem o propósito de criar ou alterar recursos no servidor.

Geralmente essas informações são enviadas através de formulários dentro da aplicação. “Um formulário HTML é um documento que contém itens que um usuário precisa fornecer. A informação é codificada em uma URL para ser transmitida para outro documento.” (COMER, 2007, p.505). Dessa maneira os dados inseridos e enviados pelo usuário são anexados a URL caso utilize o método GET ou no corpo da mensagem caso o formulário utilize o método POST.

[1] RCF2616, Documento que apresenta os padrões de funcionamento do protocolo HTTP. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. Acesso em: 14 de out. 2010.

1.2.4 Linguagens de programação

As linguagens de programação possuem um papel importantíssimo na Internet. São elas que possibilitam a abstração do mundo real em lógica computacional. Dessa maneira, é possível desenvolver sistemas para realizar tarefas. Após todo o processo necessário para a criação de uma aplicação, basta converter a lógica em código através da linguagem de programação escolhida. Existem inúmeras linguagens para desenvolvimento *web.* De acordo com a necessidade e demanda da aplicação e que a linguagem deve ser escolhida. Algumas linguagens clássicas e muito conhecidas são: Java, .Net, PHP, *Ruby on Rails, JavaScript*, *Python* e *Perl*.

1.3 Segurança

O desenvolvimento de aplicações *web* teve um grande crescimento com a evolução da Internet. Além desse crescimento, muitos sistemas legados tiveram que ser reestruturados, atualizados ou, até mesmo, refeitos, para acompanhar o ritmo atual. Com o aprimoramento, aumento da produtividade e a eficiência, as aplicações *web* ganham votos de confiança e fazem com que a sociedade crie uma forte dependência de seus serviços.

Com a responsabilidade de armazenar e disponibilizar serviços e informações, as aplicações *web* necessitam ser seguras. É justamente essa a preocupação quando tratamos de aplicações que residem na Internet. Se não houver segurança, as informações e mesmo a aplicação estão sujeitas a vulnerabilidades e técnicas de exploração como: força bruta, injeção de SQL, inserção de código malicioso etc.

Para conseguir segurança nas aplicações *web,* os desenvolvedores podem utilizar ferramentas e métodos de desenvolvimentos que garantem que a aplicação será segura, mas tudo tem em vista o custo benefício da solução. (BRAZ, 2008).

1.3.1 Fundamentos

A segurança tem o conceito de prover serviços necessários diante de situações extremas, por esse motivo tem um grande valor nos sistemas *web*:

> A segurança é vista como uma qualidade de serviço que garante o fornecimento do serviço, mesmo diante de ações de indivíduos não autorizados no sistema, sem que ocorram violações de segurança. (MELLO *et al*. Segurança em Serviços *Web,* 2006, p. 10).

A segurança é fundamentada nos seguintes itens:

1. Confidencialidade: As informações e serviços estão protegidos contra acessos não autorizados, garantido que exista autenticidade.
2. Integridade: As informações e serviços não serão perdidos ou corrompidos.
3. Disponibilidade: As informações e serviços serão acessíveis, isto é, estarão disponíveis a qualquer instante de tempo.

Esses itens devem ser seguidos a fim de prevenir possíveis falhas. Essas falhas abrem margem para a exploração de vulnerabilidades, as quais podem comprometer o funcionamento da aplicação. (LEHTINEN; RUSSELL; GANGEMI, 2006).

Conforme Mello (2006) para facilitar o entendimento de alguns eventos, quando tratamos de segurança os seguintes conceitos devem ficar claros:

1. Ameaça: ação que se realizada pode gerar consequências graves ao sistema.
2. Ataque: execução de uma ameaça, utilizando de alguma vulnerabilidade existente no sistema. O ataque possui mais quatro níveis de conceitos: (i) interceptação: em alguma parte da transição de informações existe a revelação de informações não autorizadas; (ii) interrupção: a transição de informações é interrompida, fazendo com que o destino não as receba; (iii) modificação: durante o tráfego dos dados existe a alteração por alguma parte não autorizada, e as envia para o servidor; (iv) personificação: algum indivíduo ou entidade se passa por alguma identidade autêntica na rede.

A única maneira de evitar ameaças é a correção das vulnerabilidades e o constante monitoramento das aplicações, o que torna mais complicado em sistemas complexos e de grande porte. Nesse contexto, a segurança pode ser aplicada em três linhas: física, gerencial e lógica. Nelas respectivamente temos a preocupação em proteger o meio físico (controle de acessos, reações a desastres etc.), o controle da estrutura organizacional dos processos e as próprias políticas de segurança e, finalmente, as definições de proteção que os sistemas implementarão, como controle de acesso a áreas específicas da aplicação, entre outras. (MELLO, 2006).

1.3.2 SSL

Desenvolvido pela Netscape[2], o protocolo SSL fornece confidencialidade, integridade e autenticidade para a camada de aplicação da arquitetura TCP/IP. Esse protocolo é muito utilizado quando há a necessidade de trafegar informação de maneira segura entre cliente e servidor. Dessa maneira quando existe o uso do HTTP com o SSL chamamos de HTTPS (HTTP *over* SSL).

Segundo Solomon (2005), para prover a segurança o SSL trabalha com base na troca de certificados digitais entre os sistemas. Para ter garantia, o servidor gera um certificado digital para que o cliente forneça uma chave pública e assim o servidor valide a identidade. Após fazer a verificação a comunição entre ambos é realizada através de mensagens criptografadas. Nessa comunicação, utiliza-se o protocolo HTTPS, pois ele faz com que a troca de informações seja criptografada em ambos os lados da comunicação. Essa segurança faz com que as mensagens estejam protegidas caso algum interceptador consiga capturar o tráfego entre o cliente e o servidor. Além de estarem criptografadas, para que se tenha acesso as informações contidas no tráfego, é necessário ter a chave para fazer a lógica reversa sobre o dado criptografado.

1.3.3 Criptografia

Para garantir os três itens de segurança (integridade, confiabilidade e autenticidade), existem diversas ferramentas que são destinadas a várias áreas de conhecimento e que ajudam proteger as aplicações e informações.

O primeiro nível de segurança deve partir de dentro da própria aplicação. “A segurança de software diz respeito ao uso de controles embutidos nos próprios programas, que operam independentemente das medidas de proteção a que estão submetidas as redes” (BEAL, 2005, p.98).

Para cumprir a difícil tarefa de prover segurança, foram desenvolvidas novas tecnologias e outras foram aprimoradas. A criptografia é uma técnica antiga e muito utilizada. Segundo o Instituto Nacional de Tecnologia da Informação (ITI), a criptografia significa:

[2] É importante mencionar que a Nestscape encerrou e teve suas atividades tendo seus produtos continuados. (Nota do autor). “Empresa americana fabricante de navegadores e servidores utilizados na Internet”. (SAWAYA, 1999, p.313)

“arte de escrever em códigos de forma a esconder a informação na forma de um texto incompreensível.” O ITI ainda diz que existem dois tipos de criptografia simétrica e chave pública. A simétrica utiliza o processo de cifragem e decifragem que compreende respectivamente, em codificar ou ocultar a informação e obter a informação original através da informação cifrada. Esses dois processos utilizam uma chave em comum, que deve ser compartilhada e armazenada de forma segura, pois através dessa chave é possível chegar a informação original. Já a que utiliza chave pública, ao invés de possuir somente uma chave em comum, possui duas chaves distintas a privada e a chave pública. Elas possuem uma lógica entre si, o que permite a recuperação das informações criptografadas utilizando qualquer uma.

1.3.4 Aplicações práticas

Através dessas técnicas, surgem funcionalidades tecnológicas que viabilizam a segurança na *web.* A assinatura digital utiliza da mesma funcionalidade do método de chave pública, mas opera juntamente com a função *hash.* O *hash* garante que cada assinatura terá um identificador único (*código gerado pela função hash)*, o que possibilita validar e autenticar usuários que possuam essa assinatura. O certificado digital é um documento eletrônico assinado digitalmente, ele associa pessoas ou entidades com a chave pública. A fim de aumentar a proteção das chaves, foram criados os cartões inteligentes (*smart cards)*. Eles possuem um microprocessador embutido juntamente com uma memória para armazenar informações. Dessa maneira, o próprio trabalho proteger a informação com métodos de criptografia, pode ser realizado dentro do próprio cartão de acordo com o ITI. Atualmente, já existe o e-CPF para validação de pessoas físicas, o e-CNPJ para pessoas jurídicas, entre outros. Com essas ferramentas é possível realizar tarefas com mais segurança. Porém, não existe a isenção de que esses dados sejam roubados por invasões ou tentativas de exploração de vulnerabilidades como afirma o ITI.

Capítulo 2 – Técnicas de exploração

Qualquer aplicação deve prover segurança aos seus dados e serviços, garantido os três itens básicos: confiabilidade, integridade e autenticidade. Esse é o cenário ideal, mas nem toda aplicação *web* segue esses conceitos, o que permite o surgimento de vulnerabilidades que, se exploradas, podem causar um grande prejuízo.

Para a exploração dessas brechas de segurança, os invasores utilizam de técnicas específicas para tornar uma falha uma ponte de acesso a aplicação *web*. O problema é maior quando no desenvolvimento da aplicação não houve a preocupação em validar as entradas de informações. Dessa maneira, existem várias técnicas que podem se aproveitar disso e causar grandes problemas como afirma Hoglund (2006):

> Invasores hábeis podem criar o próprio software cliente ou modificar um cliente existente. Um invasor pode (e irá) criar um software cliente personalizado capaz de enviar entradas malformadas de propósito e no momento certo. É dessa forma que se desfia o tecido da confiança. (HOGLUND, 2006, p. 43)

Este capítulo aborda as quatro principais técnicas utilizadas para a exploração de vulnerabilidades em aplicações *web*. A seção 2.1 explica o funcionamento da técnica denominada força bruta, além de citar métodos que previnem esse tipo de ataque. A seção 2.2 explica a técnica conhecida por URL *Manipulation* e apresenta exemplos acadêmicos. A seção 2.3 explica o SQL *Injection,* que é utilizado para exploração de vulnerabilidades entre a aplicação e o seu banco de dados. A seção 2.4 aborda o XSS *Cross Site Scripting*, explica suas consequências e apresenta exemplos.

2.1 Força Bruta

Nenhum sistema *web* está livre de alguma tentativa de invasão e, se apresentar falhas, a probabilidade de sucesso em uma invasão aumenta muito. Um dos métodos mais antigos e utilizados atualmente é o método de força bruta (*brute force*). Esse método consiste basicamente na tentativa de descobrir alguma informação válida através de combinações de caracteres e os utilizar como credenciais na aplicação até que obtenha sucesso. Por esse motivo também é conhecido como método de tentativa e erro.

Esse método implementa algoritmos que, além de automatizar o processo de combinações, realizam as tentativas de se autenticar na aplicação. Cada algoritmo é desenvolvido especificamente para as aplicações, pois elas podem diferir quanto ao tamanho ou tipo de informações solicitadas. “Há ferramentas de força bruta para o ataque de senha no *underground* que pode fazer centenas ou até mesmo milhares de *logins* baseados em dicionários por segundo.” (HOGLUND, 2006, p.179).

O ataque de força bruta por ter de percorrer um imenso espaço de tentativas é um método custoso e, na maioria dos casos, demorado. Existem algumas técnicas que agilizam esse processo. Ao invés do processamento ficar centralizado em um único hardware, é dispersado em vários equipamentos. Dessa maneira a velocidade com que o algoritmo realiza as tentativas aumenta consideravelmente. “O ataque pode levar horas ou até mesmo dias até que a conta seja quebrada.” (HOGLUND, 2006, p.179).

> O padrão de projeto algorítimico baseado em força bruta é uma técnica poderosa para o projeto de algoritmos quando se tem algo a procurar ou quando se deseja otimizar alguma função. Aplicando esta técnica em uma situação genérica, tipicamente enumeram-se todas as possíveis configurações das entradas envolvidas e escolhe-se a melhor das configurações enumeradas. (GOODRICH, 2007, p. 477).

Em virtude do grande poder que esse método possui, foram criadas formas de defesa nas aplicações para dificultar a sua utilização e garantir a autenticidade das informações durante o processo de autenticação.

2.2 URL Manipulation

As solicitações de recursos na Internet podem ser realizadas através de um URL, se for utilizado o método GET. Quando realizamos a solicitação de algum endereço através de um navegador informamos a URL e, em seguida, recebemos a resposta. Quando estamos em alguma aplicação que trafega os parâmetros e dados via URL, certamente é utilizado o método GET. Em alguns casos é útil e traz agilidade. Em contrapartida, pode expôr a aplicação, possibilitando o surgimento de vulnerabilidades.

Suponha uma aplicação que utiliza o método GET para transportar informações. Quando é solicitada a listagem de clientes que possuam a idade igual a 20 anos o sistema gera uma requisição no seguinte formato:

```
www.exemplo.com.br/listagem?idade=20
```

Para um usuário do sistema que o conheça bem, pode ser mais produtivo fazer a edição via URL ao invés de acionar formulários e botões para realizar uma nova pesquisa. Fazendo o bom uso da manipulação de URL (URL *Manipulation*) não haveria riscos à aplicação. Contudo, a grande preocupação é que nem todos tem boa intenção. Dessa maneira, esse método se torna perigoso e muito eficaz para se obter informações e acessos indevidos. "Um atacante pode manipular a partir do parâmetro do número de conta e passivelmente pode ver e modificar todas as contas." (SANTOS JUNIOR , 2008, p. 141).

Muitas aplicações realizam referências a objetos dentro da aplicação e normalmente estão ligadas a algum banco de dados contendo informações. Através de algumas poucas tentativas, um invasor pode realizar solicitações maliciosas que podem retornar dados altamente confidenciais.

> Este tipo de ataque aconteceu no site da Australian Taxation Office's GST Start Up Assistance em 2000, onde um usuário legítimo, mas hostil, simplesmente modificou o ABN (identificador de empresa) presente na URL. O usuário se apossou de cerca de 17.000 registros de empresas cadastradas no sistema e então enviou para cada uma das 17.000 empresas detalhes do ataque. (SANTOS JUNIOR , 2008, p. 141).

Essa técnica pode ser utilizada não apenas para recuperar informações, mas também para acessar áreas restritas dentro da aplicação, realizando assim listagem de diretórios, descoberta da estrutura da aplicação e localização de arquivos.

Segue o exemplo de um usuário mal intencionado que possui acesso a aplicação, e realiza uma alteração da URL a fim de acessar dados referentes a outro usuário. A URL original do usuário 1 seria:

```
www.exemplo.com.br/usuarios/tarefas.php?usuario=usuario1
```

Intencionalmente, o usuário realiza uma alteração em sua URL a fim de descobrir as tarefas do usuário 2:

```
www.exemplo.com.br/usuarios/tarefas.php?usuario=usuario2
```

Se aplicação não possuir controles e implementações de segurança, facilmente esses dados estão disponíveis a qualquer um que tenha acesso, quebrando assim os três itens básicos que deveriam ser garantidos a fim de se obter segurança.

2.3 SQL injection

Uma aplicação *web* possui o seu funcionamento baseado na arquitetura cliente-servidor. Logo, é válido entender que por trás de toda aplicação existe um ambiente de infraestrutura onde se encontram servidores provendo banco de dados, hospedagem de páginas *web* etc. Com a evolução que a Internet vem sofrendo, as aplicações *web* estão se tornando complexas e extensas, necessitando de gerenciamento e manutenções para que possam prover seus serviços e informações com segurança, principalmente porque possuem abrangência por toda a rede mundial.

O SQL[3] *injection* (injeção de SQL), consiste na inserção de código SQL (*Structured Query Language* ou Linguagem de Consulta Estruturada) nas entradas de dados da aplicação. Esse código é executado na aplicação atingindo camadas inferiores chegando até o banco de dados. Dessa maneira, se a aplicação não possuir mecanismos de validação e segurança, essas instruções maliciosas serão executadas e podem retornar informações, permitir acessos e, até mesmo, ocasionar paradas na aplicação. "A injeção de SQL é um ataque em que código SQL é inserido ou concatenado nos parâmetros de entrada do usuário na aplicação que são repassados para o servidor SQL para análise e execução." (CLARCKE, 2009, p. 6) (Tradução nossa).[4]

O ataque de SQL *injection* geralmente ocorre na camada de apresentação da aplicação, onde são solicitados dados aos usuários. Por isso a aplicação deve estar preparada para validar e testar todas as possíveis entradas. Caso não exista essa preparação, existirá uma vulnerabilidade que possui grande potencial de exploração, pois esse ataque visa recuperar informações ou garantir acesso a aplicação através de solicitações realizadas.

A linguagem SQL consiste em uma sequência lógica de instruções que geram inúmeros resultados. Quando uma aplicação é desenvolvida sem que haja a preocupação com a segurança na entrada de dados, é imaginado que todo o processo de inserção de dados seguirá o fluxo normal. Eis o descuido do qual o SQL *injection* se aproveita para atacar as aplicações.

Toda aplicação *web* disponibiliza informações e, para que essas informações sejam recuperadas devem existir procedimentos que façam a solicitação das informações

[3] Linguagem utilizada para acesso a dados. Através dessa linguagem é possível realizar diversas tarefas em banco de dados como, selecionar um conjunto de registros com ou sem critérios de pesquisa, alterar informações, inserir informações, excluir registros etc. (BATTISTI, 2005).

[4] SQL injection is an attack in which SQL code is inserted or append into application/user input parameters that are later passed to a back-end SQL server for parsing and execution.

desejadas. Quando é solicitada uma requisição via método POST ou método GET utilizando uma URL, a aplicação recolhe as informações e as processa para que devolva ao usuário a resposta. Esse trabalho consiste em receber os dados, validá-los, montar as instruções SQL, enviá-la ao banco de dados, receber as respostas, formatá-las e exibi-las ao usuário. A FIG. 3 apresenta a interação do usuário com a aplicação *web*.

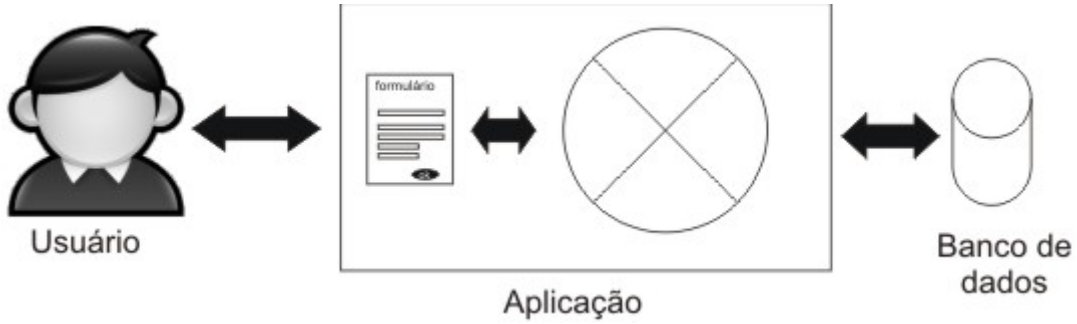

FIGURA 3 - Funcionamento básico de uma aplicação.
Fonte: Do autor.

O que ocorre é que, nem sempre, os dados são validados de maneira correta, permitindo que instruções contendo códigos maliciosos sejam enviadas a aplicação. No momento em que a aplicação necessita enviar essas solicitações ao banco de dados, é quando o SQL *injection* atua. Para demonstrar como seria realizado um possível ataque utilizando essa técnica utilizamos um exemplo de uma requisição utilizando o método GET para o envio de informações:

```
www.exemplo.com.br/func/list.php?nome=thiago
```

No exemplo acima é imaginado o fluxo correto e esperado da aplicação. Após a submissão de um formulário ela provavelmente retornaria a listagem de funcionários cujo `nome` fosse igual a `thiago`. Suponhamos que essa aplicação não esteja seguindo os itens básicos para garantir segurança. Submetendo o formulário com dados SQL concatenados ao campo `nome`, podemos simular um ataque:

```
www.exemplo.com.br/func/list.php?nome=thiago'or '1'='1
```

Essa instrução retornaria todos os nomes existentes no banco não respeitando o parâmetro de limitação (ser igual a `thiago`) . Quando essa injeção é realizada na aplicação seria gerada uma instrução SQL com o seguinte formato:

```
SELECT * FROM funcionarios WHERE nome = 'thiago' or '1'='1';
```

Nessa instrução o comando `SELECT` solicita a seleção de dados, onde o `*` representa todos os campos existentes na tabela `funcionarios` e o filtro de registros é realizado pelo comando `WHERE` possuindo a limitação através dos valores recebidos da aplicação na variável `nome`, que com a injeção ao invés de receber somente o valor `thiago`, recebeu `'thiago' or '1'='1'`. A interpretação dessa instrução pelo banco de dados é simples, pois o operador `or` (ou) exclui a necessidade da validação de ambas comparações, basta que uma esteja correta para que toda a instrução seja válida. Nesse exemplo, é mostrada uma simples listagem através do ataque e que pode ter suas consequências elevadas de acordo com a formulação e utilização de instruções de injeção.

> Existe uma infinidade de maneiras para explorar vulnerabilidades de SQL *injection*. O sucesso do ataque depende do conhecimento da base de dados e do sistema que existe por trás da aplicação. Em alguns casos, é desejável que se tenha habilidade e perseverança para conseguir explorar todo o potencial de uma vulnerabilidade. (CLARCKE, 2009, p. 8) (Tradução nossa).[5]

A exploração de aplicações utilizando a injeção de SQL, requer habilidade para poder entender, mapear toda a aplicação e a sua estrutura interior. Para utilizar essa técnica, o invasor deve ter conhecimento da linguagem SQL e das particularidades que podem ser exploradas. No exemplo citado anteriormente, a instrução parte do princípio que não existe validações na aplicação e utiliza a injeção através de combinações de instruções como caractere de escape. Dentro do SQL, todo parâmetro textual deve ser passado dentro de um par de aspas simples, o que a injeção pode fazer é inserir códigos dentro desse par. É nesse contexto que a instrução `nome = 'thiago' or '1'='1'` se aplica, pois dentro do código é esperado um valor para que seja inserido dentro de um par de aspas e, dentro desse local, o invasor aproveita para inserir vários parâmetros de forma que instrução seja aceita e válida.

Sem o tratamento necessário das informações, é possível até conseguir acesso a aplicação. Isso é perigoso, pois é possível manipular o banco de dados e até mesmo o sistema de arquivos do servidor, através de exclusão, criação e alteração de registros, tabelas e usuários. Isso mostra a força que esse tipo de ataque possui e as grandes consequências que uma tentativa de invasão bem sucedida pode fazer.

---

[5] There are many ways to exploit SQL injection vulnerabilities to achieve a myruad of goals; the success of the attack is usually highly dependent on the underlying database and interconnected systems that are under attack. Sometimes it can take a great of skill and perseverance to exploit a vulnerability to its full potential.

2.4 Cross site scripting (XSS)

A técnica *Cross site scripting*, muito conhecida como XSS, consiste basicamente em inserções de códigos HTML maliciosos na aplicação. Essa técnica é explorada quando a aplicação interage com informações oriundas do usuário e não realiza as devidas validações para sua correta exibição no navegador. A linguagem mais utilizada nessa técnica é o JavaScript[6], mas não impedindo o uso de qualquer outra linguagem *script* que seja suportada pelos navegadores.

> A técnica do XSS se caracteriza pela inserção de *tags* HTML, em particular a *tag* <script>. Quando a aplicação executa a *tag* no navegador cliente em vez de apenas imprimir este conteúdo, os dados da sessão – incluindo informações potencialmente sensíveis do usuário - podem ser enviados para qualquer destino sem que o servidor ou cliente percebam. (SANTOS, 2009, p. 67).

O XSS permite uma série de tarefas que prejudicam a aplicação, como a execução de *scripts* maliciosos no navegador da vítima, roubo de sessões, alterar a aparência de *sites*, reter o controle do navegador do usuário etc.

De acordo com Stock (2007), todos os *frameworks*[7] utilizados no desenvolvimento de aplicações *web* são vulneráveis ao XSS e que existem três tipos de XSS: refletido, armazenado e inserção DOM (*Document Object Model* ou Modelo de objeto de documento). O XSS refletido é baseado no nível mais fácil de exploração, pois ele tem como premissa que a página refletirá o dado enviado pelo usuário como retorno direto a ele. Seguindo esse raciocínio, todas as informações fornecidas pelo usuário pode ser repassadas para outros fins sem que seja percebido. Já o XSS armazenado tem por padrão o armazenamento do dado malicioso em arquivo, banco de dados e, em um certo estágio revela os dados ao usuário. Aplicações como redes sociais, lojas online, que possuam grande tráfego de informações, são o cenário ideal para utilizar o XSS.

Essa técnica se bem aplicada tem um potencial muito forte, pois ela tem a capacidade de alterar a apresentação de páginas da aplicação, manipular aspectos internos, recuperar informações, como usuário e senha, para autenticação na aplicação, ajudar no roubo de sessões etc.

> Os ataques são frequentemente implementados em Java Script, que é uma linguagem poderosa de *scripting*. O uso do Javascript habilita o atacante a manipular qualquer

[6] JavaScript é uma linguagem de *script* utilizada na programação *web*. Ela permite a interação entre a aplicação e o usuário através da criação de conteúdo HTML dinamicamente. (FLANAGAN, 2002).
[7] Estrutura que contém bibliotecas, implementações, modelos de código para desenvolvimento e organização de aplicações. (Nota do autor)

> aspecto da página a ser renderizada, incluindo a adição de novos elementos (como um espaço para *login* que encaminha credenciais para um *site* hostil), a manipulação de qualquer aspecto interno do DOM e a remoção ou modificação de forma de apresentação da página. (STOCK, 2007, p. 8).

Para exemplificar como seria um simples uso de XSS em uma aplicação desprotegida, segue uma solicitação passada via método GET através de uma URL:

```
http://www.exemplo.com.br/usuarios.php
?id=<script>alert('Exemplo de XSS');</script>
```

No exemplo acima observa-se que a variável id receberá uma instrução em Java Script para executar uma janela no navegador com a mensagem: "`Exemplo de XSS`", essa é uma utilização simples da técnica XSS.

Para conseguir ser tão poderoso, o XSS utiliza da falta de validação existente no código da aplicação ao realizar qualquer instrução sem validação, imaginando sempre que as entradas de dados serão sempre conforme esperadas. Uma vez que o XSS é implantado na aplicação, ele pode ser utilizado para enviar *emails* e informações de usuários. Isso é muito interessante, pois o usuário confiando na aplicação, qualquer mensagem oriunda da mesma não será descartada. Assim se ela trouxer um *link*, a probabilidade do usuário acessá-lo é muito alta. E para garantir sua integridade, o usuário é redirecionado para a página da aplicação com o código malicioso embutido. Isso faz com que a técnica se camufle por trás da aplicação causando sérios problemas.

Atualmente, o XSS causa problemas em aplicações de redes sociais, *home banking*, fóruns, comércios virtuais etc. De acordo com Villas Boas (2009), o *Twitter* sofreu ataques que ocasionaram o roubo de informações e senhas de usuários da aplicação utilizando a técnica de envio de informações maliciosas via a própria aplicação. Isso ocorre, pois esses ambientes são ideais para aplicar essa técnica, uma vez que existe um grande fluxo de informações, dados de outros usuários que pode ser acessíveis. Convém mencionar que grandes empresas já sofreram ataques através dessa técnica.

Capítulo 3 – Aplicabilidade

As técnicas de exploração possuem um poder muito grande. Se a aplicação não prover segurança essas técnicas podem ser utilizadas para explorar a aplicação de forma prejudicial. Da mesma maneira as técnicas podem ser aplicadas para testar se a aplicação é segura e se baseia nos três itens básicos da segurança.

O grande problema quando se pensa em segurança de aplicações *web* é que durante o desenvolvimento não são utilizadas medidas para evitar que ocorram ataques provenientes do uso dessas técnicas. As aplicações na grande, maioria só são protegidas contra os ataques mais conhecidos. Diante desse cenário, qualquer usuário mal intencionado e que possua um grau de conhecimento mais avançado certamente utilizará não uma técnica, mas uma combinação de técnicas para atingir os seus objetivos.

A fim de proteger as aplicações *web* desses ataques é necessário entender o funcionamento de cada técnica em uma aplicação. Esse capítulo aborda na seção 3.1 a utilização de cada técnica e apresenta alguns exemplos. Já na seção 3.2, é apresentado o estudo de casos de uma aplicação e mostrado exemplos ocorridos em aplicações *web*.

3.1 Utilização por técnica

3.1.1 Força Bruta

A técnica força bruta baseia-se no método da tentativa e erro. Na aplicação dessa técnica geralmente são utilizados algoritmos para automatizar as tentativas em uma aplicação. Essa aplicação pode levar horas ou até mesmo dias até se obter um resultado válido.

A força bruta é utilizada quando se deseja descobrir alguma informação válida dentro de uma aplicação como usuários, senhas, arquivos, pastas, endereços *web* etc. A sua utilização é simples, mas custosa porque ela precisa percorrer um imenso espaço de combinações, o que torna esse método um pouco improdutivo.

Para aplicar essa técnica deve-se analisar o ambiente que será utilizado e observar se não existem mecanismos que impeçam várias tentativas seguidas ou que necessitem a

validação de caracteres provenientes da percepção humana. A FIG. 4, lustra o exemplo de uma aplicação que pode ser submetida a técnica de força bruta.

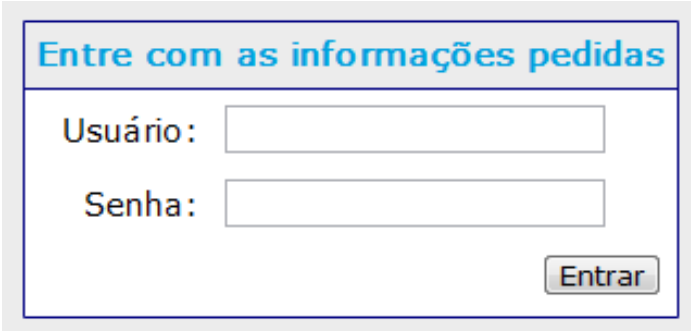

FIGURA 4 - Aplicação passível do uso da força bruta.
Fonte: Do autor.

Geralmente para utilizar essa técnica são criados algoritmos que automatizam as tentativas de inserção de dados. Como existem inúmeras variações utiliza-se essa técnica. Ela fará milhares de testes com as possíveis variações de usuário e senha. A FIG. 5 mostra um possível teste de submissão de informações a aplicação.

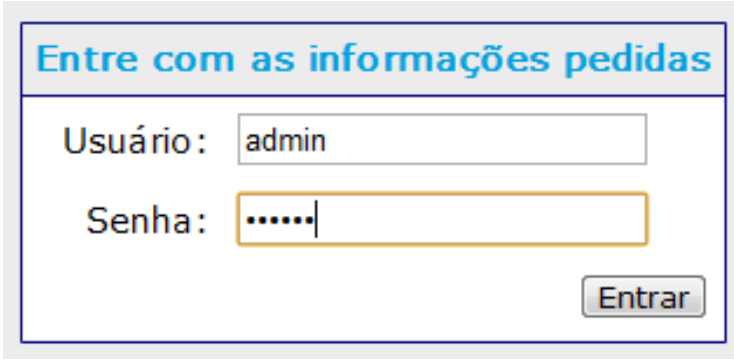

FIGURA 5 - Inserção de dados provenientes com algoritmo de força bruta.
Fonte: Do autor.

No algoritmo de força bruta existe uma área de código que tenta usuários que são muito utilizados em aplicações o *admin* e suas variações: administrador, *administrator*, adm etc. De posse desses usuários o algoritmo realiza combinações para encontrar se existe alguma senha válida para esse usuário. Após realizar inúmeros testes o algoritmo abandona esse usuário e reinicia o processo com outro. Depois de testar todos os possíveis usuários pré-cadastrados e não obter sucesso o algoritmo pode através de alguma lógica pode criar os seus próprios usuários para submeter a aplicação. Por precisar percorrer um grande espaço de possibilidades a solução com essa técnica pode demorar muito tempo e não retornar nenhuma

informação válida. Caso o usuário mal intencionado possua algum dado válido na aplicação o seu campo de tentativas diminui muito. Por exemplo se ele souber o usuário, tentará somente descobrir a senha, o que com certeza será bem mais eficiente do que combinar usuário e senha em um grande espaço de possibilidades. Um exemplo que ficou conhecido em todo o mundo foi a invasão de várias contas inclusive a conta do presidente dos EUA (Estados Unidos da América) no *Twitter*, como afirma IKEDA (2010):

> François Cousteix, 24, é um dos mais novos funcionários de uma empresa belga especializada na capitalização de audiências em redes sociais. Mas há quatro meses, sua história era bem diferente: desempregado e conhecido na *web* como "Hacker Croll", Cousteix foi detido na França após invadir uma dezena de contas do Twitter, entre elas as do presidente dos EUA, Barack Obama, da cantora Britney Spears e de Evan Williams, co-fundador e CEO do próprio Twitter. Julgado e condenado a uma pena branda – cinco meses de liberdade vigiada – Cousteix revelou por e-mail ao UOL Tecnologia os bastidores do ataque às contas. A técnica usada por Cousteix mostra como é fácil descobrir senhas na internet, desde se tenha tempo para a cansativa tarefa de adivinhá-las por tentativa e erro (e, depois de ler como ele as descobriu, é bem provável que você mude seus próprios hábitos na criação de senhas na *web*). [...] Depois de passar semanas coletando dados pessoais informados em redes sociais usadas por funcionários do Twitter, o jovem francês conseguiu adivinhar senhas frágeis de suas contas de e-mail. A partir daí, teve acesso a dados confidenciais em e-mails corporativos e ao sistema interno de gerenciamento do microblog. Para provar a falha de segurança, o "Hacker Croll" copiou as páginas de celebridades de dentro do sistema do Twitter – sem ter acesso, no entanto, a senhas dos perfis – e divulgou em fóruns e chats de tecnologia. (IKEDA, 2010).

A segurança em qualquer aplicação que manipule dados de usuários deve seguir os itens básicos para garantir a confiabilidade, integridade e autenticidade das informações. Dessa maneira há um ambiente seguro e pouco suscetível a falhas.

3.1.2 URL Manipulation

A manipulação de URL é uma técnica que permite a exploração de falhas em aplicações *web*. É muito utilizada para alterar o funcionamento correto de uma aplicação através de requisições contendo dados maliciosos. O uso dessa técnica pode fornecer ao utilizador acesso a áreas restritas da aplicação e acesso a dados teoricamente protegidos. Para aplicar essa técnica é necessário que exista um conhecimento razoável sobre o funcionamento da aplicação que será o alvo. A manipulação de URL pode ser utilizada em conjunto com outras técnicas a fim de atingir os objetivos do ataque.

Se não existir segurança na aplicação, uma vez conhecida a sua estrutura de funcionamento o uso dessa técnica não é complicado. Suponhamos que exista uma aplicação para controlar os gastos financeiros de seus usuários, e que cada usuário tenha seu *login* e senha para efetuar a autenticação na aplicação. E que o uso dessas credenciais só permita acesso a área respectiva de cada usuário. A FIG. 6 a seguir mostra a tela de autenticação para o sistema de controle financeiro.

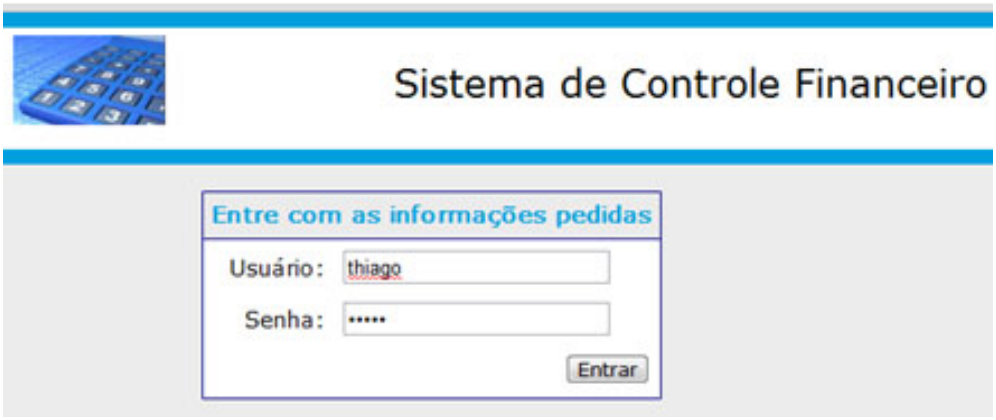

FIGURA 6 - Tela de autenticação do sistema de controle financeiro.

Fonte: Do autor.

Após a autenticação o usuário é exibida a tela do sistema conforme pode ser observado na FIG. 7.

FIGURA 7 - Tela respectiva a cada usuário do sistema.

Fonte: Do autor.

Como pode ser observado na aplicação em questão, o tráfego de informações é realizado através do método GET. Esse tráfego é notado pelas informações contidas na URL da aplicação: `http://sistemacf.com.br/app/user.php?thiago`.

Caso a aplicação não esteja preparada para uma entrada de dados inválida ela pode permitir o acesso a áreas restritas ou até ocasionar sua parada. A FIG. 8 simula a entrada de dados inesperados pela aplicação e a sua reação após a submissão.

FIGURA 8 - Inserção de dados inválidos na aplicação através da manipulação de URL.

Fonte: Do autor.

Como a aplicação não possui nenhum mecanismo para tratamento de dados, o usuário autenticado foi redirecionado para uma área administrativa, representada pela FIG. 9.

FIGURA 9 - Redirecionamento indevido após a inserção de dados não validados.

Fonte: Do autor.

Por meio da utilização dessa técnica é possível explorar a estrutura de diretórios por trás da aplicação. Isso é possível se não houver algum método que bloqueie esse ataque. A FIG. 10 mostra a implementação da falta de proteção que ocasiona a listagem de diretórios.

FIGURA 10 - Listagem de diretório indevida em uma aplicação *web*.

Fonte: Do autor.

A manipulação de URL permite que a aplicação seja explorada. Se existirem falhas o seu comportamento pode ser inesperado. Ao desenvolver uma aplicação devem ser efetuados testes de manipulação a fim de garantir a segurança e integridade da aplicação e dos dados existentes armazenados.

3.1.3 SQL Injection

SQL *Injection* é sem dúvida uma das técnicas mais utilizadas para exploração de vulnerabilidades. Isso é embasado no poder que possui e como pode ser utilizado. Por ser uma técnica que lida diretamente com a base de dados é muito comum a exploração de dados restritos em aplicações. Mesmo com esse risco, existem inúmeras aplicações que não possuem estruturas de segurança que as protejam contra esse tipo de ataque.

A aplicação dessa técnica requer conhecimentos sobre a linguagem SQL e do funcionamento da aplicação. Uma vez descoberta alguma brecha de segurança que permita a utilização dessa técnica, a aplicação estará vulnerável. O *site* da empresa de telefonia Telefônica foi invadida por um programador utilizando SQL *Injection*.

> KMax publicou dados parciais de clientes da Telefônica em um site provisório que, segundo seu advogado, funcionaria como uma prova de conceito. Recurso utilizado por hackers para demonstrar a existência dos problemas. Em entrevista à INFO Online, o hacker afirmou que vulnerabilidade explorada comprometia o banco de dados da empresa. "O site estava vulnerável à SQL Injection, falha que permite que alguém possa ganhar acesso total ao banco de dados, usando nada além do que o próprio navegador. A vulnerabilidade não existia apenas em um único ponto da página, mas em várias outras partes" **.** O problema foi corrigido pela Telefônica em 24 horas após a exposição do problema na rede. (DELLA VALLE, 2009).

Além do conhecimento da linguagem SQL é necessário saber onde e como utilizar essa técnica. Isso é realizado através de um estudo da aplicação que será o alvo do ataque. A FIG. 11 simula um ataque através dessa técnica, em que o invasor utiliza a forma mais simplificada do ataque.

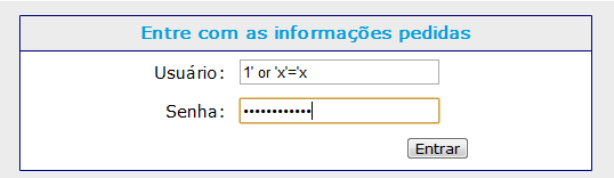

FIGURA 11 - Simulação de uma utilização do SQL Injection.

Fonte: Do autor.

O invasor imagina que a aplicação fará uma consulta ao banco de dados e que essa consulta será interpretada da seguinte maneira:

```
SELECT usuario, senha FROM tabela WHERE usuario='1' or 'x'='x'
AND senha='1' or 'x'='x';
```

Se a aplicação não disponibilizar nenhum método para filtrar as informações inseridas, essa requisição será repassada ao banco de dados. Ao receber a requisição o banco de dados não encontrará nenhum usuário e senha informado, mas validará a expressão através

da estrutura de comparação `or` que será interpretada como verdadeira conforme pode ser observado na FIG. 12.

```
SELECT usuario, senha FROM tabela WHERE
(FALSE)  or  (TRUE) =  (FALSE)  or  (TRUE) =
usuario='1' or 'x'='x'  AND senha='1' or 'x'='x';
```

FIGURA 12 - Interpretação do código de ataque.

Fonte: Do autor.

Ao utilizar o SQL Injection o atacante deve imaginar como a aplicação se comportará durante as requisições. Os erros enviados da aplicação ajudam muito na identificação do banco de dados utilizado e do ambiente da aplicação. Essas informações podem ser úteis na exploração das vulnerabilidades e ajudam muito no ataque. Por exemplo, a FIG. 13 apresenta a tela de uma aplicação *web* ao reportar um erro. Observe que é possível descobrir o banco de dados utilizado (número 1 na FIG.13), é possível saber qual o sistema operacional onde aplicação está hospedada (número 2 na FIG. 13), além de descobrir qual a versão do *framework* utilizado (número 3 na FIG. 13) e algumas informações sobre o código da aplicação conforme apresenta o número 4 na FIG. 13.

FIGURA 13 - Erros reportados pela aplicação *web* da empresa telefônica VIVO.

Fonte: Do autor.

Além de garantir acesso a aplicação o SQL Injection permite que o invasor crie usuários, delegue permissões, exclua dados, modifique a estrutura do banco etc. Se o usuário conhecer a estrutura de alguma tabela da aplicação e se ela não estiver protegida é possível realizar uma inserção no banco de dados. A FIG. 14 apresenta um campo na aplicação que está sujeito a aplicação da técnica. Nessa figura o usuário já encontra autenticado e utilizará do sua posição para realizar a exploração.

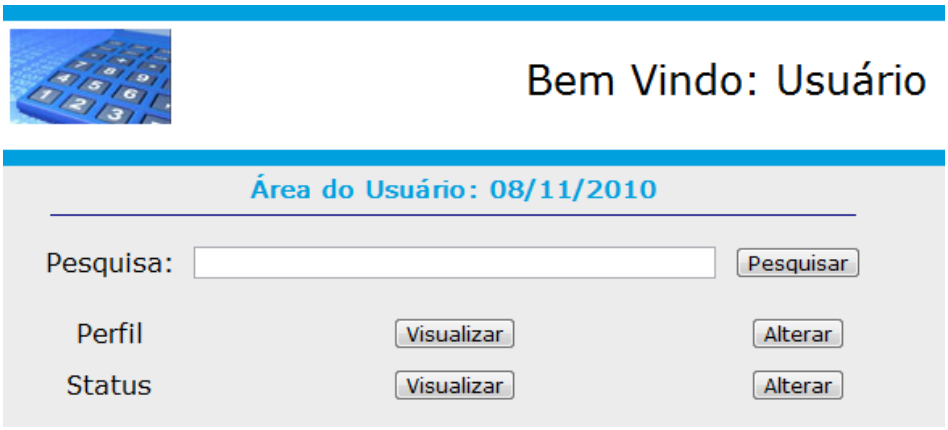

FIGURA 14 - Campo que pode ser utilizado para aplicação da técnica: Pesquisa.

Fonte: Do autor.

Se o usuário conhecer a estrutura das tabelas por trás da aplicação é possível realizar até inserções dentro do banco de dados e as utilizar posteriormente. Essa inserção poderia ser realizada se no campo de pesquisa fosse submetido a seguinte pesquisa.

```
x'; INSERT INTO usuarios ('usuario', 'senha','nome')
  VALUES ('thiago,'123456','inserção de dados');';
```

Nesse exemplo a aplicação faria a busca do dado `x` e realizaria uma inserção na tabela de `usuarios`. Por ser uma técnica muito versátil e aplicável, as aplicações devem se proteger adequadamente. A realização de testes e boas práticas de programação são a única maneira de evitar esses ataques.

3.1.4 XSS Cross Site Scripting

Através do XSS é possível inserir código malicioso na aplicação sem a percepção dos usuários. O XSS consegue se camuflar por trás da aplicação simulando um ambiente totalmente seguro e sem contratempos.

A utilização do XSS em aplicações requer o estudo a fundo da aplicação. E quanto maior for esse conhecimento mais sofisticado é o nível do ataque. Essa técnica pode ser utilizada para recuperar dados ou prejudicar o funcionamento da aplicação. A técnica se camufla por trás da aplicação fica transparente para o usuário que nem imagina o que ocorre. Uma utilização muito comum dessa técnica ocorre em *sites* de relacionamento onde o usuário tem a possibilidade de inserção de dados na aplicação. Para exemplificar o uso do XSS simularemos um ambiente de um fórum de discussão onde os usuários podem trocar informações. A FIG. 14 representa a inserção de dados indevidos em uma página de informações compartilhadas para todos os usuários do fórum.

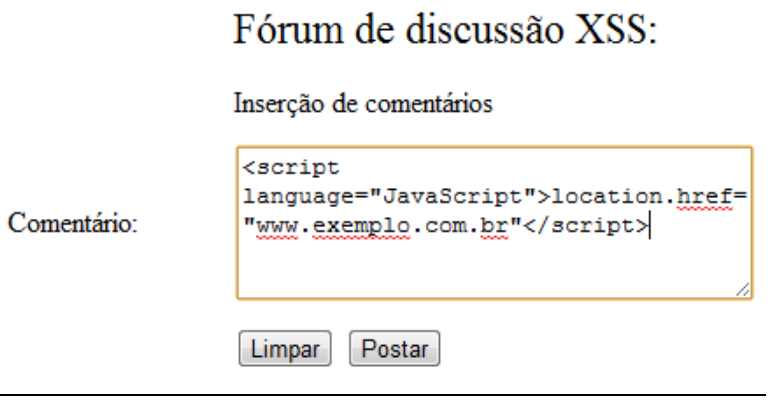

FIGURA 15 - Utilização da técnica em uma aplicação.
Fonte: Do autor.

Após enviar essas informações a aplicação, qualquer usuário que efetuar *login* na aplicação e acessar essa área na aplicação será redirecionado para a página `www.exemplo.com.br`. Esse redirecionamento poderia ser realizado para uma página que solicitasse dados pessoais dos usuários, e como foi um redirecionamento oriundo da aplicação o usuário não notaria o ocorrido. A FIG. 16 representa a página que contém código malicioso inserido pelo o usuário.

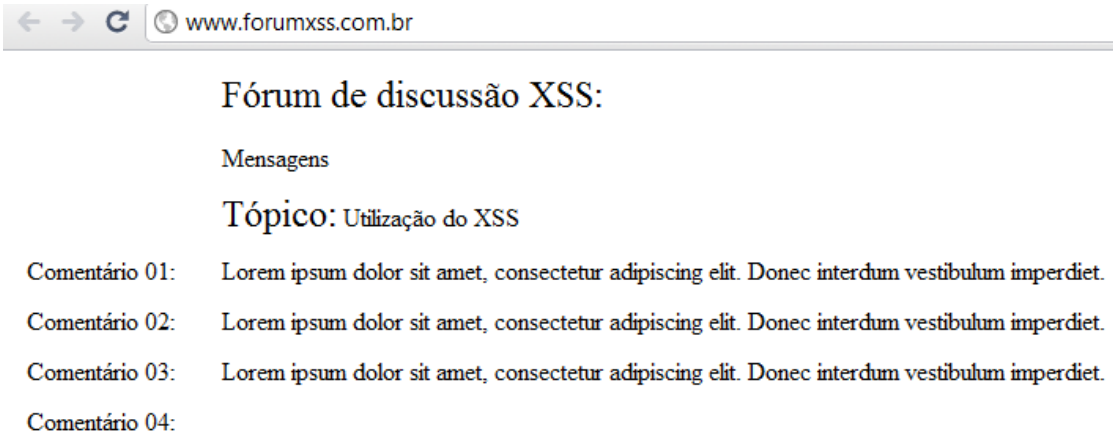

FIGURA 16 - Página infectada por código malicioso.

Fonte: Do autor.

Após alguns instantes o usuário que entra na aplicação é redirecionado a outra página em função do código malicioso inserido pelo usuário. A FIG. 17 apresenta o redirecionamento após a autenticação na aplicação.

FIGURA 17 - Redirecionamento indevido por código malicioso.

Fonte: Do autor.

As falhas de segurança do XSS são muito exploradas em aplicações *web*. Exemplos reais aconteceram com grandes redes sociais e causaram grandes problemas. Através do XSS foi possível propagar mensagens dentro de aplicações sem que os usuários tivessem conhecimento.

> Quando há uma falha de XSS, ela é interessante porque permite ao invasor incluir código no contexto daquele site. No caso do Twitter, uma falha de XSS permite que códigos sejam inseridos para realizar tarefas dentro do serviço de microblog – como, por exemplo, o envio de outros tweets. Falhas de Cross-site Scripting são fáceis de evitar. Elas existem porque o desenvolvedor do site não fez alguma verificação no conteúdo enviado pelo internauta. (ROHR, 2010).

Para evitar esses ataques a aplicação deve basear nos princípios básicos de segurança garantindo a integridade das informações de seus usuários e o seu bom funcionamento.

3.2 Estudo de casos

As vulnerabilidades encontradas em aplicações *web* são conhecidas e exploradas desde o surgimento dessa tecnologia. Grandes fatos históricos já aconteceram e retratam bem o prejuízo que a exploração dessas vulnerabilidades pode causar. O *site* da UOL realizou uma seleção de dez ataques *hackers* que causaram muitos problemas. Os ataques selecionados mostram gastos financeiros em torno de 1,4 milhões de dólares com a finalidade de tornar seguro sistemas que haviam sido invadidos. Esses ataques tinham a intenção de prejudicar vários órgãos importantes como NASA, Pentágono, Marinha dos EUA etc. O prejuízo financeiro é estimável, mas, o prejuízo moral é incalculável.

> Na ocasião, um grupo de hackers chineses chamados "Honker Union" desconfiguraram o site do Departamento de Serviços Humanos e Saúde, do Departamento de Pesquisa Geológica e da NASA. [...] Em 2006, o romeno Victor Faur foi indiciado em Los Angeles por conspiração e de tentar, por nove vezes, invadir computadores. Faur é acusado de hackear 150 computadores da NASA, do Departamento de Energia e da Marinha Americana. A façanha obrigou o departamento americano a gastar cerca de US$ 1,4 milhões para reformulação dos sistemas. (UOL, 2009).

Para saber se a aplicação está segura, é necessário realizar testes utilizando técnicas exploração. Dessa forma, é possível entender o comportamento da aplicação em diversas situações e identificar as possíveis falhas.

3.2.1 Twitter

Na rede social Twitter foram encontradas falhas graves de segurança. Essas falhas permitiam que usuários da rede social enviassem código malicioso para a aplicação. Esse código replicava mensagens para todos os contatos do usuário simulando um efeito cascata.

A técnica mais utilizada para afetar o Twitter foi o XSS. Atualmente, essa falhas foram corrigidas e não permitem a inserção de código na aplicação. A FIG. 18 exemplifica

um código utilizado na exploração dessa falha. Esse código redireciona o usuário a outra página caso ele passe o ponteiro do mouse sobre a mensgem.

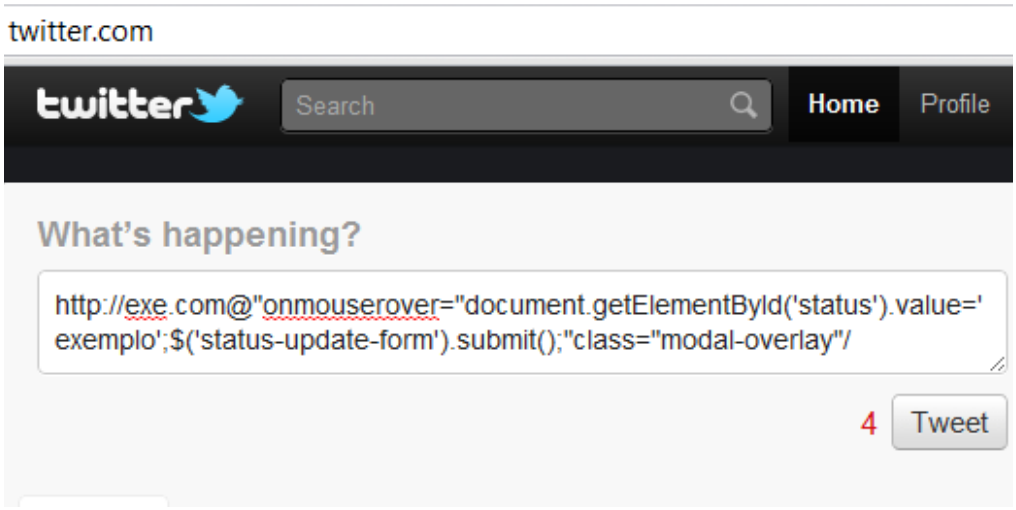

FIGURA 18 - Inserção de código malicioso em aplicações *web.*
Fonte: Do autor.

3.2.2 Universidade Católica de Goiás

No portal da universidade é disponibilizada uma área para que os alunos tenham acesso a sua área conforme pode ser observado na FIG. 19. Após a análise inicial da aplicação não é possível encontrar mecanismos para evitar ataque de força bruta e SQL *Injection.* Se algum invasor desenvolver algum algoritmo que faça milhares de tentativas simultâneas a partir de vários locais diferentes a aplicação pode não aguentar a carga e ficar indisponível. Em algumas partes do portal pode ser possível a utilização de SQL *Injection* já que não existem validações quanto a entrada de dados na aplicação.

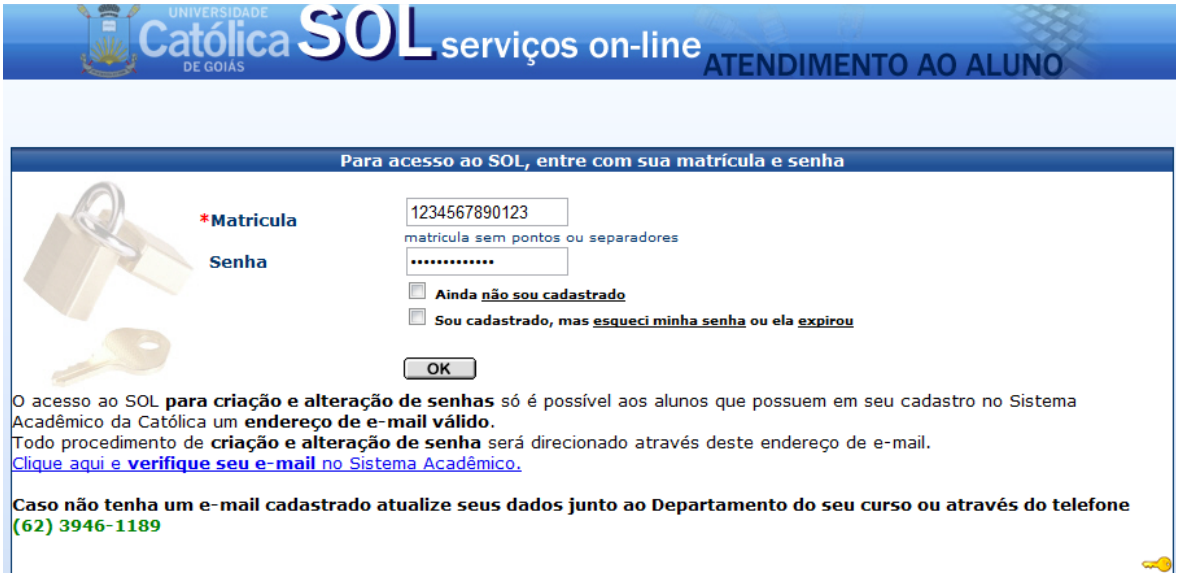

FIGURA 19 - Tela de autenticação da aplicação.

Fonte: Do autor.

Caso ocorra alguma invasão na aplicação seria possível ter acesso a vários dados restritos como endereços, emails, telefones etc. Seria incalculável a proporção dos problemas que esse ataque poderia ocasionar.

3.3 Estudo de um ataque

Para a utilização de técnicas de exploração, o invasor faz um passeio por todo o ambiente da aplicação alvo observando os possíveis pontos de falhas. Para descobri-los ele testa se aplicação valida entrada de dados, se protege os campos de busca e se possui mecanismos de segurança contra ataques.

Após essa análise, é iniciado o processo de exploração na aplicação. Esse processo constitui em aplicar técnicas para ver como a aplicação se comportará. Quando a aplicação não está preparada, ela pode retornar mensagens reportando qual a plataforma, banco de dados e a linguagem de programação da aplicação. Com essas informações, o invasor pode explorar falhas conhecidas para plataformas específicas facilitando o seu trabalho.

Na maior parte dos ataques, o invasor utiliza a combinação de várias técnicas. Ele pode se autenticar na aplicação através de ataques de força bruta e utilizar o SQL *Injection* para recuperar informações do banco de dados.

O invasor, quando tem o objetivo de invadir e prejudicar, tentará de todos os modos possíveis. E quando o ataque for descoberto, já pode ter passado muito tempo e o prejuízo ser incalculável.

Capítulo 4 – Formas de proteção

Com a tendência mundial de aumentar o número de aplicações *web* para gerenciar serviços e disponibilizar informações é necessário que exista segurança nesses ambientes.

Diante disso este capítulo apresenta algumas maneiras importantes que garantem o mínimo de segurança que uma aplicação deve fornecer. Essas técnicas partem desde o cuidado no desenvolvimento das aplicações até a atenção com o ambiente onde a aplicação será hospedada.

Para garantir os três itens básicos de segurança, a aplicação deve ser planejada e desenvolvida com atenção. A segurança deve ser implementada no início de qualquer projeto de uma aplicação. A segurança é garantida por um somatório de fatores. Esses fatores partem desde a utilização de *frameworks* no desenvolvimento até a utilização de tecnologias que garantam a autenticidade das informações inseridas na aplicação, além de contar com a segurança do ambiente escolhido para hospedar a aplicação.

As formas de proteção são divididas por técnicas, as quais são apresentadas como a seguir. Na seção 4.1 é apresentada formas de proteção a nível de ambiente. Na a seção 4.2 são apresentadas formas de proteção por técnica que são elencadas da seguinte maneira: 4.2.1 formas de proteção contra força bruta; 4.2.2 formas de proteção contra Url *Manipulation*; 4.2.3 formas de proteção contra SQL *Injection*; 4.2.4 formas de proteção contra XSS *Cross Site Scripting*.

4.1 Proteção a nível de ambiente

O primeiro nível onde é necessário que exista segurança é o ambiente onde a aplicação será hospedada. Esse ambiente constitui a plataforma do sistema operacional que será utilizado e os serviços que esse ambiente disponibilizará. É recomendável que o planejamento, preparação e instalação do ambiente sejam realizados com muito cuidado e sempre seguindo as melhores práticas contidas na documentação do fabricante. Dessa maneira é possível garantir que o sistema que hospedará a aplicação estará bem configurado e que a probabilidade de existir alguma vulnerabilidade é baixa. Para garantir a disponibilidade dos serviços é necessário que exista um acompanhamento de todo ambiente de modo que ele sempre fique atualizado. Como citado por CAMPOS (2010), frequentemente os fabricantes de

sistema soltam versões com correções de falhas em seus sistemas, ‘a Red Hat, por outro lado, no momento do fechamento deste post, já avaliou seus Enterprise Linux e concluiu que só o RHCE5 está vulnerável. A empresa planeja atualizá-lo assim que concluir testes.

Outro ponto chave da segurança no ambiente diz respeito a configuração dos serviços que o ambiente disponibiliza. Para disponibilizar uma aplicação *web* é necessário que exista um servidor *web.* O servidor *web* para o sistema operacional nada mais é que um processo que executa sobre sua plataforma e que ficará encarregado de interpretar e disponibilizar as páginas que compõe a aplicação. Para prover segurança e funcionar corretamente ele precisa ser configurado. Mesmo após realizar as devidas configurações existem técnicas que ajudam a blindar o ambiente contra tentativas de ataques. Uma vulnerabilidade muito conhecida nesse cenário é a listagem de diretórios que existem em uma aplicação. Isso pode ser facilmente protegido pela inclusão de um arquivo *index* seguido de sua extensão e com conteúdo vazio. Essa prática faz com que o diretório não seja listado e se caso seja forçada a visualização do diretório será retornada uma página em branco.

Podemos citar a questão de privilégios de acesso que os processos e usuários terão dentro do sistema operacional. O ideal é que eles tenham somente o nível básico de privilégios para executar suas tarefas. Dessa maneira é possível prevenir ataques provenientes de falhas no próprio ambiente.

Infelizmente a garantia de segurança no ambiente de qualquer aplicação requer uma constante atualização e acompanhamento. E se isso não for seguido a chance de exploração de alguma vulnerabilidade em função de um sistema desatualizado é grande. E nesse patamar a aplicação *web* seria isenta de qualquer culpa, mas prejudicada com essa falha.

4.2 Formas de proteção por técnica

Quando a segurança deve estar na aplicação vários fatores devem ser levados em consideração. Falhas na aplicação geram grandes problemas e na maioria dos casos só é percebida após alguma consequência grave. Existem várias técnicas de exploração de vulnerabilidades em aplicações *web* esse trabalho apresentou algumas das mais conhecidas e utilizadas. É válido lembrar que a combinação de várias técnicas em um ataque pode ter um poder muito grande.

Cada técnica tem o seu ponto chave de atuação e é nessa linha de pensamento que demonstraremos as formas de proteção específicas de cada técnica.

4.2.1 Métodos de proteção contra força bruta

Para garantir a proteção contra ataques de força bruta foram desenvolvidas técnicas que não permitem a total automatização do processo de autenticação e consulta a dados. Como o método de força bruta é baseado no método de tentativa e erro, nada mais cômodo e inteligente do que a automatização dessa tarefa através de um algoritmo que executará até que obtenha sucesso. Como o algoritmo é constituído por uma lógica programada, não possui a inteligência perceptiva. Levando em consideração esse ponto, foi desenvolvida a técnica de reconhecimento de caracteres em imagens geradas aleatoriamente.

No processo de autenticação, os desenvolvedores perceberam que se as entradas da aplicação fossem somente informações como usuário e senha, facilitaria muito a utilização de algoritmos, dessa maneira uma alternativa de segurança implantada é a geração de caracteres aleatórios no momento do *login*. Geralmente, esses caracteres são gerados através de imagens distorcidas fazendo com que somente a percepção humana consiga descobrir quais caracteres presentes na imagem. Conforme pode ser observado na FIG. 20 além do usuário e senha, é solicitada entrada dos caracteres gerados pelo gerador aleatório, que nesse exemplo é `3x3mplo`.

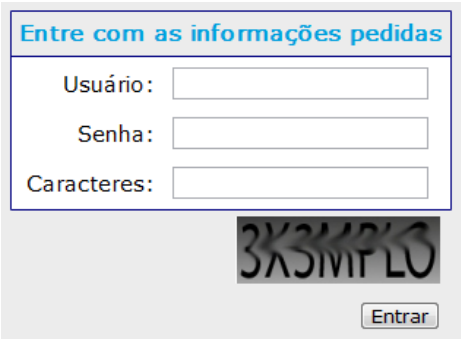

FIGURA 20 - Exemplo de *login* com validador de caracteres gerados por imagens.

Fonte: Do autor.

Outra medida adotada é não informar especificamente qual dado inserido na autenticação está errado. Em outras palavras se na autenticação somente o usuário estiver

correto e a senha errada, a aplicação deve retornar um alerta semelhante a esse: “Usuário e/ou senha inválidos” conforme pode ser observado na FIG. 21.

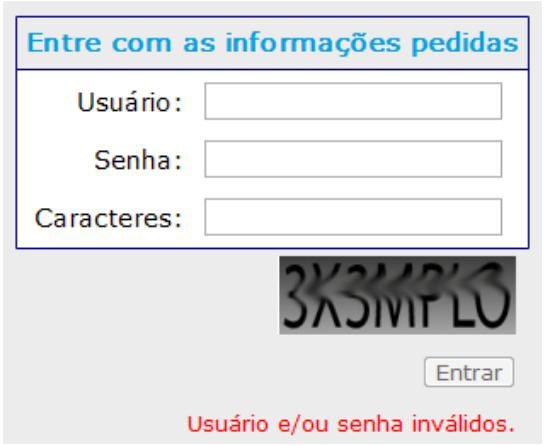

FIGURA 21 - Exemplo de mensagem genérica alertando a entrada errada de dados.

Fonte: Do autor.

Dessa forma não há como o invasor saber quais dados estão certos ou errados. Existe ainda uma alternativa muito interessante que consiste em limitar ou bloquear o número de tentativas de autenticação por intervalos de tempo. Por exemplo, a cada cinco tentativas erradas, o *login* fica bloqueado por 15 minutos conforme pode ser observado na FIG. 22.

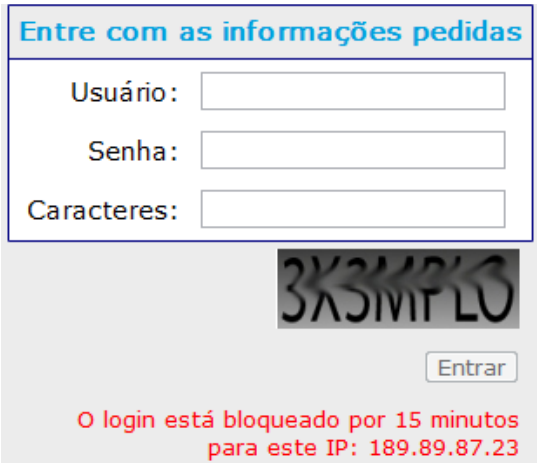

FIGURA 22 - Bloqueio de *login* para um endereço IP após diversas tentativas de acesso.

Fonte: Do autor.

A alternativa de bloqueio de *login* por intervalos de tempo impossibilita que invasores façam vários testes de credenciais em intervalos muito curtos. Assim o tempo de

tentativas aumenta tornando improdutivo o método de força bruta. Essa alternativa também pode identificar e bloquear solicitações do endereço IP que está realizando várias tentativas.

4.2.2 Métodos de proteção contra Url Manipulation

Durante o desenvolvimento de uma aplicação *web* são definidos vários aspectos de seu funcionamento. Dentre eles existe a forma como os dados são trafegados dentro da aplicação. Quando utilizado o método GET, a aplicação fica mais vulnerável a uma tentativa de exploração, pois os dados são passados via URL podendo sofrer alterações por qualquer indivíduo que tenha acesso a aplicação. Para garantir mais de segurança a aplicação e dificultar o uso da URL *Manipulation* é aconselhável desabilitar o método GET utilizar do método POST no tráfego de dados. Através desse método as informações são enviadas dentro do corpo da mensagem, e não mais na URL. Mesmo com as informações embutidas no corpo da mensagem existem maneiras de manipular o seu conteúdo e para evitar esse problema utiliza-se a criptografia das informações trafegadas. Dessa maneira, se as mensagens forem interceptadas não serão legíveis, pois necessitarão de uma chave para que haja a sua decodificação.

Na utilização dessa técnica, pode existir a intenção de acesso a diretórios da aplicação através da manipulação da URL. Existem ações que podem ser executadas no ambiente da aplicação para a proteção contra esses acessos além de implementações no código da aplicação para limitar o acesso a áreas restritas através de regras de acesso. Nesse caso ao acessar uma área imprópria o usuário seria redirecionado a uma parte pré-estipulada na aplicação.

4.2.3 Métodos de proteção contra SQL Injection

A técnica SQL Injection tem por base a exploração de vulnerabilidades em aplicações *web* com a finalidade de recuperar informações contidas no banco de dados. A forma mais eficiente de evitar que essas falhas aconteçam é seguir boas práticas de programação durante o desenvolvimento da aplicação.

O SQL Injection aproveita da falta de validação da entrada de dados na aplicação para recuperar informações restritas. Assim, uma das maneiras de proteger uma aplicação contra ataques utilizando o SQL *Injection* é validar a entrada de dados da aplicação. Essa validação não permitiria a entrada de caracteres especiais como aspas, parênteses, operadores matemáticos etc. Em uma autenticação que necessite informar usuário e senha é de extrema importância a realização de verificações dos dados informados.

Outra forma de se proteger é utilizar funcionalidades do banco de dados que já implementam segurança para o recebimento de parâmetros, as API's (*Application Programming Interfaces*). Uma API contém rotinas padrões para execução de tarefas e já implementam a segurança necessária para evitar ataques dessa natureza.

Outro ponto chave é garantir que o usuário de sua aplicação não tenha privilégio de acesso aos dados de outras aplicações evitando que ataques a aplicação prejudique outras aplicações que compartilhem o mesmo servidor de banco de dados.

4.2.4 Métodos de proteção contra XSS Cross Site Scripting

O XSS tem por finalidade a inserção de código malicioso na aplicação. Esse código pode ser mascarado pela aplicação de forma que ele pareça integrado a ela. Assim como o SQL *Injection* o XSS aproveita a falta de validação na entrada de dados da aplicação para atuar. Baseado nisso, uma das maneiras de se proteger é fazer uma excelente validação da entrada de dados na aplicação evitando que caracteres especiais não sejam aceitos. Outro ponto de segurança é garantir o bom funcionamento do controle de sessão. Ela deve ter um limite de tempo quando ociosa e não permitir que seja roubada por outro usuário da aplicação.

Conclusão

Com a grande utilização das aplicações em diversos ramos de atuação é imprescindível que exista segurança para garantir a autenticidade, confiabilidade e disponibilidade dos serviços e informações.

É possível garantir a segurança em qualquer aplicação que será publicada. Para que isso se torne viável, é necessário que exista o conhecimento de conceitos básicos do funcionamento das aplicações *web* e segurança. Através desse conhecimento, é possível entender como os métodos e protocolos funcionam, além de saber quando e como utilizá-los.

A segurança deve ser aplicada desde o planejamento de qualquer aplicação, seguindo as melhores práticas. Mas, para que sejam desenvolvidas aplicações seguras, deve-se conhecer como as técnicas de exploração funcionam e atuam em uma aplicação *web.* Dessa forma, é possível desenvolver e utilizar métodos que previnam a aplicação contra esses ataques.

Para garantir que a aplicação seja segura, é interessante que sejam aplicados testes na aplicação utilizando as técnicas de exploração. Dessa maneira, é possível compreender e analisar o comportamento em diversas situações em que a aplicação possa ser submetida. A maior parte de ataques bem sucedidos são realizados através tentativas de submissão que não seguem o fluxo esperado pelos desenvolvedores, ocasionando comportamentos inesperados na aplicação.

O poder de atuação que as técnicas possuem é devastador. Quando grandes ataques ocorrem, é impossível calcular o prejuízo gerado. Mesmo com todo esse perigo é possível identificar falhas de segurança em vários sistemas existentes na Internet e que através de técnicas de exploração são invadidos e tem suas informações e serviços prejudicados. Existem vários exemplos de invasões a sistemas de grandes empresas e o tamanho do prejuízo moral e financeiro sofrido. E mesmo assim ainda existem aplicações que não se preocupam com segurança.

Existem várias formas de proteção para evitar que as aplicações fiquem vulneráveis a ataques. Mas, para que isso seja aplicado, deve haver uma conscientização que a segurança deve caminhar junto ao desenvolvimento da aplicação e não após o término. A manutenção e garantia da segurança necessita da constante atualização dos conceitos e conhecimentos sobre as tecnologias emergentes.

E baseando na responsabilidade que as aplicações carregam consigo, é impossível não pensar no quesito segurança como divisor de águas para o bom funcionamento dessas ferramentas do mundo atual e para garantir que as informações e serviços estejam seguros e disponíveis aos seus usuários.

Com grande evolução tecnológica em que vivemos é esperado que a necessidade de novas aplicações para o gerenciamento de serviços e informações aumente em um número exponencial. O que infere que a segurança será primordial para que essas aplicações se tornem viáveis e confiáveis aos seus usuários. Dessa maneira, é possível garantir existam os três princípios básicos da segurança nas aplicações disponibilizadas.

Referências

BATTISTI, Júlio. **SQL Server 2005 Administração e Desenvolvimento Curso Completo**. Rio de Janeiro: Axcel Books, 2005.

BRAZ, Fabrício. **Segurança de Aplicações**. Universidade de Brasília, 2008. Disponível em: <http://rbrito.googlecode.com/svn/diversos/UNB/GSIC700-seguranca_aplicacoes /texto_-_seguranca_em_aplicacoes.pdf>. Acesso em: 01 nov. 2010.

CAMPOS, Augusto. **Bug: Canonical lança versões corrigidas do kernel, Red Hat está analisando soluções**. BR-Linux.org, 21 set. 2010. Disponível em: <http://br-linux.org/2010/bug-canonical-lanca-versoes-corrigidas-do-kernel-red-hat-esta-analisando-solucoes/>. Acesso em: 17 nov. 2010.

COMER, Douglas E. **Redes de computadores e internet [recurso eletrônico]**: abrange transmissão de dados, ligações inter-redes, *web* e aplicações. Tradução Álvaro Strube de Lima, 4 ª Ed., Dados Eletrônicos. Porto Alegre: Bookman, 2007.

COSTA, Daniel Gouveia. **Java em rede**: recursos avançados de programação. Rio de Janeiro: Brasport, 2008.

DELLA VALLE, James. **Advogado fala sobre invasão da Telefônica**. Info Exame, 31 ago. 2009. Disponível em: <http://info.abril.com.br/noticias/seguranca/advogado-fala-sobre-invasao-da-telefonica-31082009-5.shl>. Acesso em: 01 nov. 2010.

LANAGA, David. **JavaScript o guia definitivo.** São Paulo: O'Reilly & Associates, Inc., 2002

FREEMAN, Elisabeth; FREEMAN, Eric. **Head first HTML with CSS and XHTML**. United States of America: O'Reilly Media Inc., 2005.

FOWLER, Martin. **Padrões de arquitetura de aplicações corporativas.** Tradução Acauan Fernandes. Porto Alegre: Bookman, 2006.

GOODRICH, Michael T., TAMASSIA, Roberto. **Estrutura de Dados e Algoritmos em JAVA**. 4 ed. Porto Alegre: Bookman, 2007.

HOGLUND, Greg, MACGRAW, Gary. **Como quebrar códigos: a arte de explorar (e proteger) software**. São Paulo: Pearson Makron Books, 2006.

IKEDA, Ana. **Francês que invadiu Twitter de Obama e Britney mostra como é fácil descobrir senhas**. Uol Tecnologia, 20 jul. 2010. Disponível em: <http://tecnologia.uol.com.br/seguranca/ultimas-noticias/2010/07/20/invasor-do-twitter-de-obama-revela-como-descobriu-senhas.jhtm>. Acesso em: 31 out. 2010.

KUROSE, James F., ROSS, Keith W.. **Rede de computadores e a Internet**: uma abordagem top-down.Tradução Arlete Simille Marques. Revisão Técnica Wagner Luiz Zucchi. 3. ed. – São Paulo: Pearson Addison Wesley, 2006.

LEE, Valentino; SCHNEIDER, Heather; SCHELL Robbie. **Aplicações móveis**: arquitetura, projeto e desenvolvimento. Tradução: Amaury Bentes & Deborah Rudiger; Revisão técnica: Renato Haddad. São Paulo: Pearson Education do Brasil, 2005.

LEHTINEN, Rick; RUSSELL, Deborah; GANGEMI, G. T. **Computer Security Basics**. Sebastopol, California: O`Reilly, 2006.

ROHR, Altieres. **Falhas mostram despreparo de sites de redes sociais.** G1, 27 set. 2010. Caderno Tecnologia e Games. Disponível em: <http://g1.globo.com/tecnologia /noticia/2010/09/falhas-mostram-despreparo-de-sites-de-redes-sociais.html>. Acesso em: 12 nov. 2010.

SANTOS JUNIOR, Alfredo Luiz dos. **Quem mexeu no meu sistema?** : segurança em sistemas de informação. Rio de Janeiro : Brasport, 2008.

SANTOS, Célio de Jesus, JUNIOR, Cloves Ferreira. **Linux Magazine**. A revista do profissional de TI. Belo Horizonte, ano 5, n. 60, p.66-71.

SAWAYA, Márcia Regina. **Dicionário de Informática e Internet**. São Paulo: Nobel, 1999.

SCRIMER, Rob et al.; **TCP/IP: a bíblia**. Tradução de Edson Furmankievicz, DocWare Traduções Técnicas. Rio de Janeiro: Elsevier, 2002, 6ª reimpressão.

SOLOMON, Michael G.,CHAPPLE Mike. **Information Security Illuminated**. Jones & Bartlett Learning, 2005.

STOCK, Van Der Andrew, WILLIAMS, Jeff. WICHERS, Dave. **OWASP TOP 10: As 10 vulnerabilidades de segurança mais críticas em aplicações WEB, 2007**. Disponível em: <http://www.owasp.org/images/4/42/OWASP_TOP_10_2007_PT-BR.pdf>. Acesso em: 07 out. 2010.

VILLAS BOAS, Gustavo. **Microblog Twitter é alvo constante de ataques virtuais**. Folha de São Paulo, São Paulo, 09 maio 2009. Caderno serviço.