



TÚLIO SPURI TEIXEIRA DE OLIVEIRA

**TESTES DE SEGURANÇA EM APLICAÇÕES *WEB*
SEGUNDO A METODOLOGIA *OWASP***

LAVRAS - MG

2012

TÚLIO SPURI TEIXEIRA DE OLIVEIRA

**TESTES DE SEGURANÇA EM APLICAÇÕES *WEB* SEGUNDO A
METODOLOGIA *OWASP***

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientador

Prof. Dr. Joaquim Quinteiro Uchôa

LAVRAS - MG

2012

TÚLIO SPURI TEIXEIRA DE OLIVEIRA

**TESTES DE SEGURANÇA EM APLICAÇÕES WEB SEGUNDO A
METODOLOGIA OWASP**

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

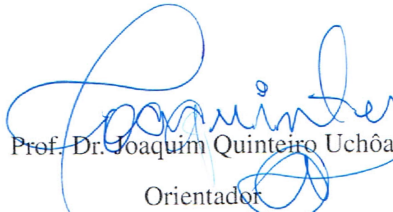
Aprovada em 09 de Outubro de 2012

Prof. Dr. José Monserrat Neto

UFLA

Prof. Dr. Raphael Winckler de Bettio

UFLA


Prof. Dr. Joaquim Quinteiro Uchôa

Orientador

LAVRAS - MG

2012

RESUMO

A Internet tem se tornando um ambiente para realização de diversas tarefas. Isto se deve ao grande uso de aplicações *Web* para tarefas como acesso a bancos, realização de compras, inscrição em eventos, compartilhamento e edição de documentos, entre outras. Desta forma este trabalho aborda a questão da segurança nas aplicações *Web*. O objetivo é avaliar a segurança empregada. Os projetos da *OWASP*, comunidade dedicada aos assuntos relacionados à segurança de aplicações *Web*, foram tomados como base para este trabalho. Para conduzir os testes foi usado o guia de testes da *OWASP – OWASP Testing Guide Versão 3.0*. As aplicações *Web* testadas foram as aplicações da categoria *e-commerce*. São elas: *e-commerce real*, *e-commerce PrestaShop* e um *e-commerce básico* implementado de acordo com as orientações sobre desenvolvimento seguro da *OWASP*. Nos testes, o *e-commerce real* foi considerado o mais inseguro, pois foi possível obter dados sensíveis de diversos clientes. As outras aplicações testadas comportaram-se de forma segura.

Palavras-chave: Aplicações *Web*; Segurança Computacional; Testes de Segurança; *OWASP (Open Web Application Security Project)*

SUMÁRIO

1	INTRODUÇÃO	10
1.1	Motivações	11
1.2	Objetivos e metodologia	12
1.3	Estrutura do texto	13
2	APLICAÇÕES WEB	15
2.1	Comentários iniciais	15
2.2	A Web	15
2.3	As aplicações.....	18
2.4	Arquitetura	20
2.5	Tecnologias.....	21
2.6	Comentários finais	27
3	SEGURANÇA COMPUTACIONAL	28
3.1	Comentários iniciais	28
3.2	Importância da segurança computacional	29
3.3	Princípios de segurança	33
3.4	Ameaças à segurança	38
3.5	Comentários finais	41
4	SEGURANÇA EM APLICAÇÕES WEB	43
4.1	Comentários iniciais	43
4.2	Vulnerabilidades em aplicações Web	44
4.3	Comentários finais	48
5	OWASP - OPEN WEB APPLICATION SECURITY PROJECT ..	49
5.1	Comentários iniciais	49
5.2	Projetos	49

5.3	<i>OWASP Testing Guide</i>	52
5.4	Comentários finais	53
6	TESTES DE SEGURANÇA EM APLICAÇÕES WEB	54
6.1	Comentários iniciais	54
6.2	Metodologia de testes	54
6.3	Coleta de informações	55
6.3.1	Identificação de <i>robots</i>	55
6.3.2	Descoberta em motores de busca	56
6.3.3	Identificação de pontos de entrada.....	57
6.3.4	Avaliação da assinatura do servidor <i>Web</i>	58
6.3.5	Análise de códigos de erros	59
6.4	Teste de autenticação	60
6.4.1	Transporte de credenciais através de um canal criptografado...	61
6.4.2	Avaliação da enumeração de usuários	62
6.4.3	Conta padrão de usuário	63
6.4.4	Teste de força bruta.....	64
6.4.5	Contorno do esquema de autenticação	65
6.4.6	Vulnerabilidades em <i>Esqueci minha senha e Lembrar senha</i>	66
6.4.7	Avaliação do mecanismo de <i>logout</i>	67
6.4.8	Avaliação do <i>CAPTCHA</i>	68
6.5	Teste de autorização	69
6.5.1	Teste de passagem de diretório	69
6.5.2	Burlagem do esquema de autorização	70
6.6	Avaliação da lógica de negócios.....	71
6.7	Testando a validação de dados.....	73
6.7.1	Testando <i>XSS</i> não-persistente	73

6.7.2	Testando XSS persistente.....	74
6.7.3	SQL Injection	76
6.7.4	XML Injection	78
6.7.5	SSI Injection	79
6.7.6	HTTP Response Splitting	80
6.8	Avaliação da negação de serviço.....	81
6.8.1	Avaliação do bloqueio de contas de usuários.....	82
6.8.2	Registro de dados do usuário no disco.....	82
6.9	Teste de gerenciamento de configuração	83
6.9.1	Testando SSL/TLS	83
6.9.2	Avaliação da infraestrutura.....	84
6.9.3	Testando o tratamento de extensões de arquivos	85
6.9.4	Arquivos sem referência, antigos e de <i>backups</i>	86
6.9.5	Testando os métodos HTTP.....	87
6.10	Teste do gerenciamento de sessão	88
6.10.1	Avaliação do esquema de gerenciamento de sessões	88
6.10.2	Avaliação dos atributos dos <i>cookies</i>	89
6.10.3	Avaliação de <i>session fixation</i>	91
6.10.4	Avaliação CSRF	91
6.11	Avaliação da técnica <i>Ajax</i>	93
6.12	Avaliação de <i>web services</i>	94
6.12.1	Avaliação do <i>WSDL</i>	95
6.12.2	Avaliação da estrutura do <i>XML</i>	95
6.12.3	Avaliação do conteúdo do <i>XML</i>	96
6.13	Comentários finais	96
7	ANÁLISE E DISCUSSÃO	97

7.1	Comentários iniciais	97
7.2	<i>E-commerce real</i>	97
7.2.1	Teste do <i>e-commerce real</i>	98
7.3	<i>E-commerce PrestaShop</i>	106
7.3.1	Teste do <i>e-commerce PrestaShop</i>	106
7.4	<i>E-commerce básico</i>	112
7.4.1	Teste do <i>e-commerce básico</i>	113
7.5	Comentários finais	117
8	CONCLUSÃO	119

LISTA DE FIGURAS

Figura 1	Troca de mensagens HTTP entre cliente e servidor	18
Figura 2	Funcionamento do CGI.....	23
Figura 3	Captura de tela do <i>WebScarab</i> após alguma navegação no <i>browser</i>	51
Figura 4	<i>E-commerce real</i> : selo indicando site blindado.....	97
Figura 5	<i>E-commerce real</i> : informações de conexão segura.....	99
Figura 6	<i>E-commerce real</i> : mensagem informando credenciais erradas	99
Figura 7	<i>E-commerce real</i> : mensagem informando a ação do mecanismo de recuperação de senhas.....	99
Figura 8	<i>E-commerce real</i> : mensagem informando a inexistência do e-mail na loja	100
Figura 9	<i>E-commerce real</i> : mensagem informando a existência do e-mail na loja	100
Figura 10	<i>E-commerce real</i> : boleto obtido através do escaner	104
Figura 11	<i>E-commerce PrestaShop</i> : dicas de segurança ao final da instalação.....	107
Figura 12	<i>E-commerce PrestaShop</i> : mensagem informando credenciais erradas.....	107
Figura 13	<i>E-commerce PrestaShop</i> : mensagem informando um erro de <i>token</i>	109
Figura 14	<i>E-commerce PrestaShop</i> : mensagem informando credenciais erradas.....	111

LISTA DE TABELAS

Tabela 1	Situações possíveis para enumeração de usuários.	62
Tabela 2	Avaliação do <i>e-commerce real</i>	105
Tabela 3	Avaliação do <i>e-commerce PrestaShop</i>	113
Tabela 4	Avaliação do <i>e-commerce básico</i>	116
Tabela 5	Avaliação do <i>e-commerce real</i> , <i>e-commerce PrestaShop</i> e <i>e-commerce básico</i>	118

1 INTRODUÇÃO

Com o crescimento da Internet, vemos cada vez mais aplicações típicas de ambientes *desktops* migrarem para o ambiente *Web*. Com esta mudança de ambiente surgiu um novo termo: *cloud computing* - computação em nuvem. Segundo (ARMBRUST *et al.*, 2010), a computação em nuvem refere-se tanto as aplicações que são entregues como serviços através da Internet quanto aos sistemas de *hardware* e *software* nos *data centers* que fornecem esses serviços.

De forma prática, pode-se citar o conjunto de aplicativos para escritório fornecida pelo Google¹, o Google *Docs*. Ele permite a criação de documentos de texto, planilhas, formulários, apresentações, entre outros, via navegador *Web*, ou seja, sem a necessidade de instalação dos aplicativos da suíte. Os dados produzidos pelo usuário são salvos nos servidores próprios do Google e podem ser acessados a partir de qualquer computador com acesso à Internet e credenciais da conta associada.

Segundo (KIM, 2009), entre as vantagens da computação em nuvem pode-se citar a despreocupação do usuário quanto aos recursos computacionais necessários às aplicações, pois todo o gerenciamento de recursos, tais como servidores, *software*, armazenamento e rede são feitos pelo provedor do serviço.

Além destas aplicações disponibilizadas no ambiente *Web* existem também os diversos *sites* de instituições e empresas que usam páginas *Web* com formulários para preenchimento de informações pessoais para algum tipo de negócio, como inscrições para vestibulares, estágios, eventos. Em todos esses casos é necessário avaliar a segurança empregada, pois se um sistema deste tipo, com diversas

¹<https://www.google.com.br/>

informações pessoais de diversas pessoas falhar, todas essas informações estarão comprometidas e podem ser usadas indevidamente.

Para podermos mensurar o perigo associado a este tipo de acontecimento, relembremos o caso do vazamento de informações que aconteceu em agosto de 2010, no qual dados de 12 milhões de inscritos no Enem - Exame Nacional do Ensino Médio, ficaram públicos na Internet. Dados como nome, CPF, RG, data de nascimento e outras informações podiam ser vistos por qualquer pessoa².

Nota-se então que este tipo de ambiente fornece muitas facilidades ao usuário final. Porém existem diversas implicações caso ocorram falhas em sistemas que lidam com informações e operações sensíveis.

1.1 Motivações

A Internet tem se popularizado cada vez mais. No Brasil, mais de 83 milhões possuem acesso à Internet³. Este crescente uso da rede mundial de computadores forneceu às empresas uma oportunidade para expandir seus negócios. O uso de lojas virtuais – *e-commerces* – também vêm se expandindo cada vez mais. O comércio eletrônico movimentou R\$ 540 milhões em 2001. Já em 2011 o valor passou para R\$ 18,7 bilhões⁴.

²A notícia “Dados de 12 milhões de inscritos no Enem vazam na internet” pode ser lida na íntegra no endereço: <http://www.abril.com.br/noticias/brasil/dados-12-milhoes-inscritos-enem-vazam-internet-584472.shtml>.

³A notícia “Brasil tem 83,4 mi de pessoas conectadas à internet” pode ler lida na íntegra no endereço <http://info.abril.com.br/noticias/internet/brasil-tem-83-4-mi-de-pessoas-conectadas-a-internet-25092012-33.shl>.

⁴Como pode ser visto na notícia “Brasil é o quinto país mais conectado do mundo” no endereço <http://info.abril.com.br/noticias/internet/brasil-e-o-quinto-pais-mais-conectado-do-mundo-22042012-7.shl>.

Além das lojas *online*, outros diversos serviços têm sido migrados para o ambiente *online*. Tornou-se possível, com o uso de aplicações *Web*, realizar transações bancárias, inscrever-se em concursos e eventos, comprar ingressos, entre outras atividades.

No âmbito acadêmico existem sistemas de gestão. A Universidade Federal de Lavras, por exemplo, possui o Sistema Integrado de Gestão – SIG⁵, que compartilha informações entre os diversos departamentos, setores e pró-reitorias. Dados de todos professores, funcionários e alunos são gerenciados por tal sistema.

Todas essas aplicações citadas lidam com diversas operações e informações sensíveis. Operações de compra envolvem nome completo, CPF, RG, número de cartões de créditos, entre outros. Desta forma tornou-se fundamental focar com mais rigor a segurança de tais aplicações. A natureza de suas operações e dados é bastante sensível. Com isso a Internet tornou-se um ambiente ideal para roubo de informações. Usuários maliciosos usam de diversas técnicas para conseguir esses dados, que posteriormente são usados para lesar as vítimas.

Enfim, a presença massiva de aplicações *Web* em diversas áreas, aliada ao crescente uso para a realização de tarefas que envolvem dados confidenciais e a necessidade de muito mais cuidado em relação a segurança das aplicações, motivaram esta pesquisa.

1.2 Objetivos e metodologia

O principal objetivo deste trabalho é avaliar a segurança de aplicações *Web* da categoria *e-commerce*. Tais aplicações são bastante usadas atualmente e lidam com

⁵<https://www.sig.ufla.br/>

operações e informações sensíveis. Desta forma, aplicações *Web* de *e-commerce* foram submetidas à testes de segurança.

As aplicações de *e-commerces* testadas em relação a segurança foram: (1) um *e-commerce* real; (2) um *e-commerce* de código aberto; e (3) um *e-commerce* básico desenvolvido segundo as orientações para desenvolvimento seguro da *OWASP*. Para avaliar a segurança de tais aplicações, foi usado o guia *OWASP Testing* Versão 3 (MEUCCI, 2008).

1.3 Estrutura do texto

O Capítulo 2 apresenta os conceitos gerais relacionados às aplicações *Web*, abordando a *Web* como plataforma de execução de aplicações e apresentando a arquitetura de aplicações *Web*, bem como algumas tecnologias que podem ser usadas para implementação de tais aplicações.

No Capítulo 3, o tema *segurança computacional* é abordado como um todo. Nesse capítulo é destacada a importância de políticas de segurança para garantir a integridade de dados sensíveis, e mostrados os danos que podem ser atingidos com uma falha de segurança. Princípios de segurança são abordados.

A seguir, o Capítulo 4 aborda o termo *segurança* com enfoque nas aplicações *Web*. As vulnerabilidades mais comuns neste tipo de aplicação são apresentadas.

O Capítulo 5 tem como objetivo apresentar a comunidade *OWASP*, seus guias, projetos e aplicações destinados à segurança de aplicações *Web*. Expõe-se também o guia utilizado neste trabalho, *OWASP Testing Guide* Versão 3 (MEUCCI, 2008).

Na sequência, o Capítulo 6 apresenta extensivamente a metodologia de testes de aplicações *Web* da *OWASP*.

Por fim, os resultados dos testes feitos em aplicações *Web* da categoria *e-commerce* são apresentados no Capítulo 7.

2 APLICAÇÕES WEB

2.1 Comentários iniciais

Este capítulo tem como objetivo apresentar uma visão geral da *Web*, apontando elementos básicos para seu funcionamento. Também serão apresentados diversas tecnologias que tornam possíveis o uso de serviços através dela. Termos correlatos serão explanados.

2.2 A *Web*

A *World Wide Web*, ou também WWW, W3 ou simplesmente *Web*, é um conjunto de documentos *onlines* (páginas) que podem conter diversas informações em diferentes formatos, normalmente texto, imagens e vídeos, e que são interligadas. Como visto em (ALONSO *et al.*, 2010; PETER, 2004), ela foi criada por Tim Berners em 1990 no *European Laboratory for Particle Physics* (CERN).

Em seus primórdios, a *Web* era uma coleção de páginas pessoais e *sites* contendo uma variedade de informações constituídas apenas de recursos estáticos. Entende-se por recurso estático algo que não muda de requisição para requisição, como um arquivo HTML ou uma imagem. Toda vez que esse tipo de recurso é requisitado, por diferentes usuários, o servidor irá responder enviando o mesmo conteúdo para cada um deles, simplesmente uma cópia. Atualmente as páginas ainda possuem diversos conteúdos estáticos, porém vários conteúdos são gerados dinamicamente após as requisições. Páginas passaram a incorporar recursos dinâmicos, os quais oferecem possibilidades de interação com usuário. Os recursos dinâmicos são gerados com base em alguns parâmetros como identidade do usuá-

rio, entrada por ele fornecida, entre outros. Esses parâmetros são manipulados por códigos no servidor, o qual irá gerar um conteúdo diferenciado para diferentes usuários.

As páginas da *Web* são implementadas usando a linguagem de marcação HTML (*HyperText Markup Language*) [Linguagem de Marcação de Hipertexto], que permite criar documentos com títulos, textos, tabelas, listas, imagens, etc. Elas possuem *links* que possibilitam carregar novas páginas. Também são constituídas de objetos que, de acordo com Kurose e Ross (2006), são simplesmente arquivos - tal como um arquivo HTML, uma imagem JPEG, um *applet*⁶ Java, ou um clipe de vídeo - que se pode acessar com um único URL. A maioria são constituídas de um arquivo-base HTML e diversos objetos referenciados.

Toda página possui uma identificação única na *Web* que é o URL (*Uniform Resource Locator*) [Localizador Padrão de Recursos] que é um conjunto de caracteres que representa um recurso disponível na Internet, como descrito na RFC 1738 (BERNERS-LEE; MASINTER; MCCAHILL, 1994).

Para que as páginas possam ser disponibilizadas na Internet é necessário um servidor para hospedá-las. Os servidores são computadores que estão na rede e que são devidamente configurados para gerenciar recursos e prover serviços a outros computadores. Ou seja, um programa cliente, em um computador X, solicita e recebe um serviço de um programa servidor, em um computador Y. A arquitetura deste tipo de interação é chamada de cliente-servidor.

Existem diversas funções que podem ser exercidas pelos servidores, assim ele executará um *software* adequado àquela finalidade. Um servidor FTP, por exem-

⁶Entende-se por *applet* uma aplicação pequena que pode ser incorporada em uma página *Web*. Este assunto é abordado na Seção 2.5

plo, executa um determinado tipo de *software* que o habilita a realizar transferências de arquivos com outro computador na Internet. Como pode ser visto em (KUROSE; ROSS, 2006), o FTP (*File Transfer Protocol*) [Protocolo de Transmissão de Arquivos] é o protocolo de rede que especifica como clientes e servidores irão se comunicar. Retornando ao contexto das páginas *Web*, o servidor *Web* é quem provê o serviço, ou melhor, fornece as páginas aos clientes. A função básica deles é atender às requisições HTTP por páginas *Web* feitas por clientes *Web*. Para tal função eles podem executar o *software Microsoft Internet Information Server*⁷ (IIS) ou *software Apache Web Server*⁸, entre outros. Os clientes *Web* são os aplicativos conhecidos como navegadores *Web*, como por exemplo *Mozilla Firefox*⁹, *Microsoft Internet Explorer*¹⁰ e *Apple Safari*¹¹, entre outros.

A solicitação de páginas feita pelos clientes ocorre de acordo com o especificado no protocolo HTTP (*HyperText Transfer Protocol*) [Protocolo de Transferência de Hipertexto]. Como visto em (KUROSE; ROSS, 2006), o HTTP é o protocolo de comunicação usado para acessar a *Web* e é usado por todas aplicações *Web* atualmente. Ele define a estrutura das mensagens que são trocadas entre cliente e servidor. A Figura 1 ilustra a troca de mensagens entre cliente e servidor.

A definição do protocolo HTTP pode ser vista na RFC 1945 (BERNERS-LEE; FIELDING; FRYSTYK, 1996) e na RFC 2616 (FIELDING *et al.*, 1999).

⁷<http://www.iis.net/>

⁸<http://httpd.apache.org/>

⁹<http://www.mozilla.org/>

¹⁰<http://windows.microsoft.com/pt-br/internet-explorer/products/ie/home>

¹¹<http://www.apple.com/br/safari/>

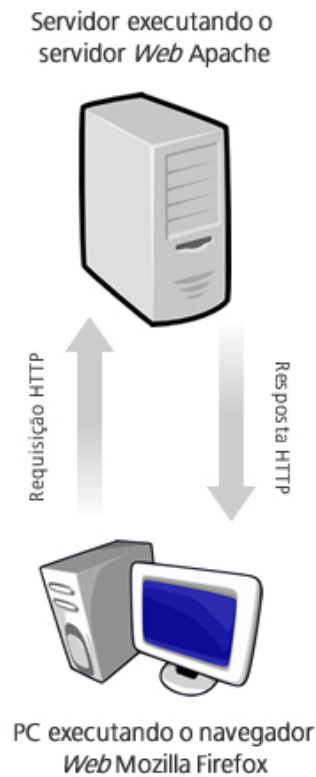


Figura 1: Troca de mensagens HTTP entre cliente e servidor

2.3 As aplicações

Como a *Web* fornece um ambiente propício às aplicações houve uma mudança de mentalidade quanto à execução de aplicações: do ambiente *desktop* para o ambiente *Web*. As aplicações *Web* têm sido desenvolvidas para realizar praticamente tudo o que for possível de ser feito no ambiente *online*. Algumas funções comuns às aplicações *Web* são: compras, leilões, apostas, bancos, buscas, *blogs* e *e-mail*. Além dessas, as aplicações *Web* têm sido usadas também para prover uma interface administrativa para diversos tipos de dispositivos, como roteadores e impressoras, por exemplo.

Existem diversas vantagens em se utilizar aplicações *Web*, abaixo algumas delas são abordadas:

Compatibilidade: como as aplicações *Web* são executadas em servidores e acessadas via navegador, a compatibilidade é bem maior se comparada aos aplicativos em *desktops*. Os desenvolvedores implementam pensando apenas no servidor alvo, ou seja, aquele que irá hospedar a aplicação. Eles não precisam preocuparem-se com os diversos sistemas operacionais, pois o que deve ser compatível com o sistema operacional é o navegador *Web* que, normalmente, não é fornecido pelos desenvolvedores da aplicação e sim por outras empresas como *Mozilla*, *Microsoft* e *Apple*.

Gerenciamento: aplicações *Web* são instaladas em um servidor e exigem poucos clientes. Isso torna a manutenção e atualização da aplicação mais simples se comparada às manutenções e atualizações de aplicações em *desktops*.

Implantação: a implantação de aplicações *Web* é simples pois todo um sistema pode ser disponibilizado apenas liberando nomes de usuários, senhas e o endereço do sistema para os funcionários de uma empresa poderem acessá-lo.

Redução de custos: aplicações *Web* podem reduzir drasticamente os custos com suporte e manutenção.

Disponibilidade: as aplicações *Web* estão disponíveis 24 horas por dia e 7 dias por semana, desde que se tenha acesso à Internet. Da mesma forma que os usuários podem acessar o *e-mail* via navegador *Web* a qualquer hora que desejar, assim também é com as aplicações *Web*.

Proteção contra pirataria: como as aplicações estão instaladas em um servidor e não nos clientes, torna-se muito mais difícil pirateá-la. Ela é gerenciada pela empresa que a implementou, assim sendo não é disponibilizado nenhum instalador para outros servidores. É importante destacar que isso depende de outros fatores como os quesitos de segurança relacionados ao servidor que hospeda a aplicação.

Com apenas alguns pontos positivos em relação às aplicações *Web* é possível notar que tal ambiente é interessante para as empresas tanto de desenvolvimento quanto para as empresas que necessitam de sistemas e que optam por sistemas *online*.

2.4 Arquitetura

As aplicações *Web* podem ser compostas por diferentes camadas. O modelo típico é a arquitetura de três camadas que é composta de camada de apresentação, camada de negócios e camada de dados (CURPHEY *et al.*, 2005).

A camada de apresentação, também chamada de *view*, tem como objetivo gerar a saída em HTML para o usuário com pouca ou nenhuma lógica da aplicação (CURPHEY *et al.*, 2005).

A camada de negócios, também chamada de lógica da aplicação ou *controller*, tem como objetivo receber a entrada do usuário e direcioná-la aos diferentes fluxos de trabalho da aplicação. É nesta camada que os dados fornecidos pelos usuários serão validados antes de serem processados pela aplicação. Assim um *controller* deve garantir que os dados estão seguros ou estão prontos para serem exibidos com segurança em uma *view* (CURPHEY *et al.*, 2005).

A camada de dados, também chamada de *model*, encapsula, isto é, agrupa funcionalidades em um mesmo contexto. Como exemplo, uma funcionalidade chamada conta, o *model* irá agrupar ações relacionadas à conta como transferência de fundos, consulta ao saldo, entre outras. Esta camada é responsável por checar se os dados estão de acordo com as regras de negócio da aplicação e também por armazenar os dados (CURPHEY *et al.*, 2005).

2.5 Tecnologias

Como já apresentado inicialmente, a *Web* era apenas uma coleção de páginas estáticas (sites) que não possibilitavam a interação com o usuário (STUTTARD; PINTO, 2007). Sendo assim, a função básica dos servidores *Web* era apenas receber a mensagem de requisição HTTP, extrair o objeto de seu dispositivo de armazenamento, encapsular o objeto em uma mensagem de resposta HTTP e então enviá-la ao cliente que solicitou (KUROSE; ROSS, 2006). Nenhum outro processamento precisava ser feito. Com o passar do tempo foram surgindo tecnologias que possibilitam a criação de páginas mais dinâmicas. Um breve histórico das tecnologias que permitem criar páginas dinâmicas é apresentado na sequência.

Em relação às tecnologias, é importante conhecer a distinção entre *script client-side* e *script server-side*. Como pode ser encontrado em (JACOBS; RAGGETT; HORS, 1999), um *script client-side* é um programa (ou porção de código) que acompanha ou é incorporado diretamente em um documento HTML. Tal código será executado na máquina do cliente, especificamente no navegador *Web*, quando o documento for carregado. O *script server-side* por sua vez é executado no servidor da página solicitada. Após execução dos códigos incorporados na pá-

gina, ela é enviada ao navegador *Web* do usuário que a requisitou (BHATNAGAR, 2008).

As tecnologias *server-side* serão apresentadas primeiramente e em seguida as *client-side*.

CGI

O CGI (*Common Gateway Interface*) foi inventado pela NCSA para o servidor *Web* NCSA HTTPd em 1993. Ele especifica um padrão para a passagem de dados entre uma página e um servidor *Web*, tal padrão pode ser consultado na RFC 3875 (ROBINSON; COAR, 2004). O CGI não é o programa em si, é a conexão (ou interface) entre a página e o servidor *Web* que permite a interação.

Quando um servidor *Web* recebe um requisição de um *script* CGI, o servidor irá executar o *script* como um outro processo, isto é, como uma aplicação à parte. O servidor passa alguns parâmetros a este processo e coleta a saída que então é retornada ao usuário via navegador *Web*. A Figura 2 ilustra o funcionamento básico do CGI.

Algumas linguagens populares para a programação usando CGI são *AppleScript*, *C/C++*, *C Shell*, *Perl*, *Tcl* e *Visual Basic*, segundo (GUNDAVARAM, 1996).

Java

A linguagem de programação Java teve seu desenvolvimento iniciado em 1990 pela *Sun Microsystems*. Ela é uma linguagem orientada a objetos e projetada para ser independente de máquina. A compilação de um código Java gera um código universal, chamado de *bytecode*, que é executado em uma máquina virtual

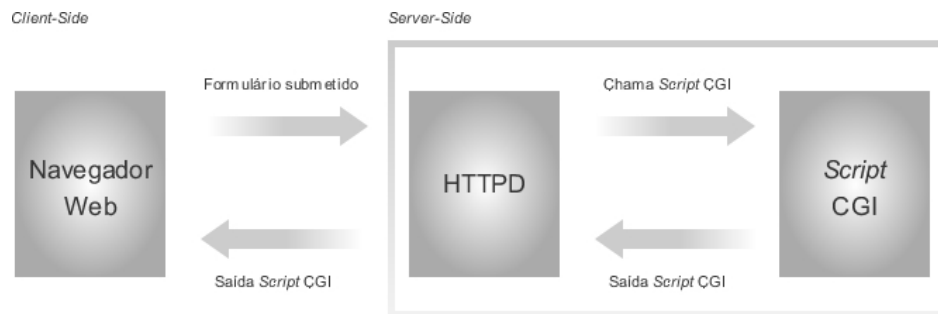


Figura 2: Funcionamento do CGI

Java. Assim um mesmo código pode ser executado em diferentes plataformas e diferentes sistemas operacionais (NIEMEYER; KNUDSEN, 2005). O Java possui algumas versões voltadas para diferentes áreas, são elas:

J2SE: *Standart Edition*, é a versão padrão do Java. Usado em aplicações *desktop* e linha de comando.

J2EE: *Enterprise Edition*, é a versão do Java voltado para Web.

J2ME: *Micro Edition*, é a versão do Java voltada para dispositivos móveis, tais como: celulares, *tablets*, *smartphones*, sistemas embarcados, entre outros.

Através de *applets*, códigos Java podem ser executados em navegadores *Web*. Entende-se por *applet* uma aplicação pequena que é incorporada em páginas *Web* usando *tags* especiais. Para o navegador, o *applet* é apenas mais um tipo de objeto a ser exibido.

Ainda em relação a Java, é possível usar *JavaServer Pages* – JSP. Como pode ser visto em (FIELDS; KOLB; BAYERN, 2001), é uma linguagem *server-side* que permite incorporar códigos Java em páginas HTML com o objetivo de dinamizá-las.

ASP.NET

A linguagem ASP (*Active Server Pages*) foi a primeira linguagem de *scripts* da *Microsoft* (CHADWICK, 2011). Posteriormente foi desenvolvido o ASP.NET que é considerada a nova geração do ASP, mas não como um *upgrade* de versão, já que foi construída do zero e não é compatível com o ASP clássico. Algumas características do ASP.NET são relacionadas em (WALTHER, 2003):

- Diferente das versões anteriores do ASP que interpretava códigos, o ASP.NET usa código compilado escrito em *Visual Basic* e *C#*, por exemplo.
- O ASP.NET é parte do *.NET Framework*, o que dá a possibilidade de uso de diversas classes *.NET* que realizam várias tarefas.

Como visto em (WALTHER, 2003), o ASP.NET é a parte do *.NET Framework* que é a infraestrutura da *Microsoft* para desenvolvimento e execução de aplicações. Ele disponibiliza diversas bibliotecas que auxiliam na construção de aplicações, é compatível com diversas linguagens de programação (*C#*, *C++*, *Visual Basic*, *J#*) e fornece ambientes de desenvolvimento como *Visual Studio .NET* e *Visual Web Developer*.

PHP

Em junho de 1995, Rasmus Lerdorf anunciou o PHP - *Personal Home Page Tools* (*PHP Tools*), assim chamado na primeira versão. No primeiro anúncio Rasmus dizia sobre o PHP: “*Essas ferramentas são um conjunto de binários CGI pequenos e coesos escritos em C*”. Em seguida ele descrevia as funcionalidades desta primeira versão. Posteriormente vieram novas versões. O PHP 3 é o que

mais reflete o que ele é hoje. Foi nesta versão que o nome foi alterado para um acrônimo recursivo: *PHP: Hypertext Preprocessor*. Na versão 5 do PHP tornou-se possível usar a orientação a objetos. Mais sobre a história do PHP pode ser encontrado em <http://www.php.net/manual/en/history.php>.

O PHP é uma linguagem de *scripts* que tinha como propósito inicial gerar conteúdos HTML dinâmicos. Porém também oferece suporte para a criação de arquivos PDFs, XMLs, imagens JPG, GIF ou PNG. É uma linguagem interpretada, ou seja, o interpretador irá ler e executar linha por linha sem gerar código objeto. O código em PHP pode ser colocado juntamente com a linguagem HTML nas páginas.

JavaScript

Outra maneira de dinamizar páginas *Web* é usar a linguagem *JavaScript*. O *JavaScript* é uma linguagem de programação *client-side* que foi criada em 1995 por Brendan Eich (GOODMAN *et al.*, 2010). Ela permite aos autores de páginas *Web* adicionar comportamentos dinâmicos em suas páginas. Um simples exemplo é exibir uma mensagem como “Bom dia!”, “Boa tarde!” ou “Boa noite!” de acordo com o horário do computador do usuário.

O *JavaScript* é também chamado de linguagem de *scripts*, como pode ser visto no site da *W3Schools*¹². Entende-se por *scripts* arquivos que contêm códigos que serão interpretados e servem para estender a funcionalidade de um programa ou neste caso de uma página *Web*. Assim sendo, os códigos escritos em *JavaScript* são inseridos diretamente nas páginas HTML. Quando a página é carregada no navegador, ele irá interpretar e executar os códigos que estão na página.

¹²<http://www.w3schools.com/js/default.asp>

Ajax

Como pode ser visto em (GARRETT *et al.*, 2005), comumente, aplicações *Web* funcionam da seguinte forma: ações realizadas pelo usuário na interface disparam requisições HTTP ao servidor *Web*, o servidor recebe a requisição, realiza algum processamento e então retorna uma página HTML para o usuário. Logo a cada requisição feita, o usuário deve ficar esperando o processamento e retorno do servidor. O *Ajax* elimina este modelo de requisitar, esperar processamento, requisitar e esperar processamento. Isto é possível pois ele introduz uma espécie de camada entre o cliente e o servidor, que é responsável por renderizar a interface e comunicar com o servidor *Web*. Desta forma ações do usuário que poderiam disparar uma requisição HTTP toma forma de uma chamada *JavaScript* ao motor do *Ajax*. Este por sua vez lida com a requisição fazendo-a assíncronamente. Neste contexto, entende-se por assíncrono a capacidade de comunicar-se com o servidor *Web* sem precisar carregar toda a página novamente.

De acordo com (GARRETT *et al.*, 2005), *Ajax* é um conjunto de tecnologias que são usadas juntas para um determinado propósito. Ele incorpora as seguintes tecnologias:

- XHTML¹³ e CSS¹⁴ para apresentação do conteúdo.
- Exibição dinâmica e interação usando DOM¹⁵.

¹³XHTML é uma sigla para *eXtensible HyperText Markup Language*. Ele é uma reformulação do HTML 4.0 (XHTML... , 2000).

¹⁴CSS é uma sigla para *Cascading Style Sheets*. É uma linguagem usada para definir estilos de páginas *Web*

¹⁵DOM é uma sigla para *Document Object Model*. O DOM é um modelo, especificado pela W3C, que permite que programas e *scripts* acessem e atualizem a estrutura e os estilos de documentos *online*. Este documento processado pode ser incorporado posteriormente ao documento original (DOCUMENT... , 2005).

- Troca e manipulação de dados usando XML¹⁶.
- Requisição de dados assíncronos usando *XMLHttpRequest*¹⁷.
- *JavaScript* ligando tudo.

2.6 Comentários finais

Atualmente no mercado tecnológico é comum encontrarmos dispositivos eletrônicos que possuem um único propósito: permitir acesso à Internet. Os *netbooks*, *tablets* e celulares em geral são alguns exemplos desses tipos de dispositivos. Estes possuem pequeno poder computacional, porém para acesso à Internet e serviços fornecidos através dela, são suficientes. Assim, com a popularização da Internet, empresas viram a possibilidade de usar tal ambiente para oferecer serviços, tais como: compras, serviços bancários, notícias, vídeos, relacionamentos e muitas outros.

Este capítulo apresentou uma visão geral da *Web*, apontando elementos básicos para seu funcionamento. Além disso foram apresentadas diversas tecnologias que tornam possíveis o uso de serviços através dela.

¹⁶XML é uma sigla para *eXtensible Markup Language*. Ele é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais, como transporte e armazenamento de dados.

¹⁷*XMLHttpRequest* é uma API, ou seja, um conjunto de funcionalidades que podem ser usadas pelo programador, que permite enviar requisições HTTP assíncronas diretamente a um servidor *Web*. A resposta do servidor pode ser usada para alterar o DOM do documento (KESTEREN, 2010).

3 SEGURANÇA COMPUTACIONAL

3.1 Comentários iniciais

Anteriormente ao uso dos sistemas computacionais para o gerenciamento e processamento de informações, a preocupação com a segurança da informação era provida através de meios físicos e administrativos. O primeiro faz uso de armários e/ou cofres robustos contando com fechaduras diversas para proteger documentos confidenciais. O segundo conta com uso de procedimentos para a seleção de pessoal (STALLINGS, 2008). Com o início dos sistemas computacionais, porém antes do crescimento e popularização da Internet, os computadores eram sistemas “isolados”, ou seja, havia pouca ou nenhuma interconectividade entre máquinas. Assim, a segurança computacional não era um aspecto tão crítico para as instituições. Porém os tempos, agora, são outros. Hoje praticamente todos os computadores - servidores, computadores pessoais, celulares, *tablets*, sistemas embutidos, entre outros - possuem alguma interconectividade.

Com a popularização e crescimento da Internet é comum cada vez mais vermos notícias relacionadas a invasões de sistemas, roubo de informações e vírus. A Internet é um ambiente hostil, assim essas notícias se tornaram frequentes nos jornais, revistas e televisão. O fato é que mesmo com tantas notícias sobre questões envolvendo sistemas computacionais e violação dos mesmos, a segurança computacional por vezes não é pensada e implementada com a seriedade que lhe é devida. Os sistemas computacionais são recursos críticos que apoiam a missão de uma instituição. Protegê-los pode ser tão crítico quanto proteger as finanças, recursos físicos ou empregados da instituição. Em suma, o propósito da segu-

rança computacional é proteger os recursos, tais como informações, *software* e *hardware*, de uma instituição/organização (GUTTMAN; ROBACK, 1995).

Como apontado por Uchôa (2005), o termo segurança computacional é relativo ao ambiente em que é utilizado. O que é seguro para uma instituição pode não ser para outra. Assim é necessário que as instituições definam o que é mais importante para o estabelecimento da segurança computacional em seu ambiente. Faz-se necessário a definição de uma política de segurança. De acordo com Soares, Lemos e Colcher (1995), uma política de segurança é um conjunto de leis, regras e práticas que regulam como uma organização gerencia, protege e distribui suas informações e recursos. Um dado sistema é considerado seguro em relação a uma política de segurança, caso garanta o cumprimento das leis, regras e práticas definidas nessa política.

3.2 Importância da segurança computacional

O mundo esta cada vez mais dependente de recursos computacionais, tanto para controle de processos industriais quanto para o entretenimento em geral. Aplicativos estão sendo usados para gerenciamento e monitoramento de processos industriais. *Video games* estão cada vez mais desenvolvidos e com funcionalidades que permitem a interação de diversos jogadores através da rede. Todos estes meios de usar sistemas computacionais trazem consigo perigos relacionados aos ciber-criminosos.

Diversas empresas têm optado por usar sistemas que utilizam *software* para monitorar e controlar remotamente processos industriais críticos, são os chamados Sistemas de Controle de Automação e Monitoramento Industrial - *Supervisory Control and Data Acquisition* - conhecidos pela sigla SCADA. São usados em pro-

cessos industriais como fabricação de produtos, geração de energia; processos de infraestrutura como tratamento e distribuição de água, oleodutos e gasodutos, sistemas de transmissão elétrica e grandes sistemas de telecomunicações (SHAW, 2006). Também para facilitar processos em construções, aeroportos, *shoppings* e estações de trem, metros, etc. Tais sistemas trazem benefícios às empresas e, pelo fato de serem totalmente baseados em sistemas computacionais, são alvos de ataques e invasões, com objetivos de roubo de informações sensíveis, caracterizando espionagem industrial, e tomada do controle dos equipamentos que eles gerenciam. Como exemplo pode-se citar as ações de cibercriminosos em companhias ligadas à energia elétrica¹⁸.

Ataques a sistemas SCADA podem afetar diversas máquinas em todo o mundo. Estes sistemas trabalham em infraestruturas de enorme abrangência, nível nacional por exemplo, assim as consequências geradas por um ataque é bem significativo. Um ataque a uma corporação de energia elétrica, por exemplo, pode afetar populações inteiras. Um exemplo de vírus criado para atacar sistemas com a tecnologia SCADA é o Stuxnet. Ele foi descoberto em junho de 2010 e foi usado para atingir organizações iranianas¹⁹.

A saída para combater esses tipos de cibercrimes é atuar em conjunto com outros países e organizações. Isto é o que vem sendo discutido atualmente em países como a Coreia do Norte, China, Estados Unidos e Coreia do Sul. Estes países resolveram se unir para criar unidades cibermilitar para proteger suas infraestrutu-

¹⁸Como pode ser visto na notícia “Cibercrime chega à infraestrutura e acende sinal vermelho na empresas de energia” no endereço: <http://convergenciadigital.uol.com.br/cgi/cgilua.exe/sys/start.htm?infoid=23953&sid=18>

¹⁹A notícia “Siemens alerta sobre vírus que ataca controles industriais” que relata a descoberta do vírus Stuxnet pode ser vista no endereço <http://computerworld.uol.com.br/seguranca/2010/07/18/virus-de-espionagem-industrial-ataca-sistemas-siemens/>.

ras e responder aos ataques, o que a Kaspersky²⁰ chama de “Interpol da Internet”, fazendo alusão à agência internacional de polícia que ajuda outras agências a localizar criminosos que agem em diversos países. A idéia é justamente obter a cooperação global das autoridades para manter os criminosos sob controle²¹.

Uma outra área que também é afetada é a área de entretenimento. A PSN - *PlayStation Network* é uma rede *online* de jogos fornecida pela Sony. Além de jogos, a PSN abriga conteúdos digitais e a loja virtual para os consoles da empresa. Em abril de 2011 ela foi alvo de ataques de *hackers*, o que resultou no bloqueio de 93 mil contas²² de usuários da rede, que poderiam ter seus dados, inclusive informações do cartão de crédito por exemplo, roubados.

Não são apenas as empresas que são alvos de ataques e invasões, os usuários comuns também são alvos. Muitas vezes são leigos no assunto e não dispõem de ferramentas para prover a proteção necessária. Além das abordagens tradicionais como o *phishing*²³ e o envio de *e-mails* falsos, eles são usados em ataques maiores sem que saibam, atuam como “zumbis”. Os computadores zumbis são máquinas controladas remotamente sem que seus usuários saibam. São utilizados por criminosos para realizar atividades ilegais e também como integrantes de ataques maiores contra instituições governamentais ou privadas. Nesta situação a maioria dos usuários não percebem que seu computador está sob controle de outra pessoa,

²⁰Kaspersky é uma empresa russa que atua na área da Segurança Computacional.

²¹A notícia “Cibercrimes: criação de ‘Interpol da Internet’ pode ser a saída” pode ser encontrada na íntegra no endereço: <http://computerworld.uol.com.br/seguranca/2011/10/07/paises-devem-se-unir-na-protacao-de-dados-sensiveis/>

²²A notícia “Sony bloqueia 93 mil contas de serviços on-line por ataque hacker” pode ser encontrada no endereço: <http://g1.globo.com/tecnologia/noticia/2011/10/sony-bloqueia-93-mil-contas-de-servicos-line-por-ataque-de-hackers.html>

²³*Phishing* é uma maneira de tentar conseguir dados como *logins*, senhas, números de cartões de créditos, etc, se passando por uma intuição verdadeira e confiável. Um exemplo seria construir uma cópia exata de um site de banco e enviar aos usuários com o argumento de renovação de conta ou similar.

já que ele continua funcionando, porém quando eles começam a atacar sites ou enviar quantidades massivas de *e-mails*, a velocidade da rede cai consideravelmente.

Pequenas mudanças dificultam ações de invasores. Usuários comuns não se preocupam com senhas fracas, conexões não criptografadas e *backups*. São pequenos detalhes que ajudam a manter os usuários com o mínimo de segurança na rede. O problema é que poucas pessoas preocupam-se até que sejam um alvo real de algum ataque ou falha. O descuido em relação a Segurança Computacional em geral foi confirmado em (UCHÔA; CAMINHAS; SANTOS, 2009), que apresenta uma consulta aos alunos de disciplinas ligadas à área no curso de Ciência da Computação da Universidade Federal de Lavras, e constatou que menos de 20% do alunos tinham feito *backup* de seus arquivos mais importantes há menos de um mês.

Ações em conjuntos podem trazer bons resultados na luta contra o cibercrime, pois como mostrado em (HOWARD; LEBLANC, 2001), existe um dilema entre as ações cabíveis a invasores e defensores. Os pontos de segurança de um determinado *software* ou processo, por exemplo, devem ser totalmente cobertos pelos defensores, ou seja, devem ser todos protegidos. Já o invasor escolhe um entre esses pontos e então o explora. É bem provável que o ponto escolhido seja o ponto mais fraco. Relacionado a este aspecto, existe a questão dos ataques e falhas no qual o defensor irá proteger aquilo que ele conhece, em contrapartida o invasor explora alguma vulnerabilidade até então desconhecida. Dentro do âmbito da lei, os defensores caminham na legalidade enquanto os invasores podem “jogar sujo”, não preocupando-se com as consequências e punições relacionadas às ações tomadas. Finalmente em relação a vigilância, os defensores devem estar sempre atentos, já os invasores podem atacar a qualquer hora, aquela em que for mais conveniente.

Este dilema nos mostra o quão desigual é o combate contra o cibercrime, ainda mais em um ambiente onde a anonimato é uma “arma” ao lado dos criminosos.

É possível observar então que as questões relacionadas à Segurança Computacional devem ser pensadas e projetadas da melhor forma possível, pois uma vulnerabilidade em um sistema computacional traz várias implicações bem como gastos à empresa alvo. De acordo com notícia veiculada em sites relacionados à segurança, os crimes cibernéticos são responsáveis por um gasto de R\$ 15,3 bilhões, só no Brasil²⁴.

Em virtude do que foi mencionado, pode-se notar o quão importante é manter seguro seja qual for o ambiente, sistemas industriais a *consoles* para entretenimento. O comprometimento de um sistema traz diversas implicações aos usuários e também as empresas envolvidas, o que pode levar a perdas e processos penais.

3.3 Princípios de segurança

Segundo Pfleeger e Pfleeger (2006), a segurança computacional é construída sob três pilares: *confidencialidade, integridade, disponibilidade*. Diversos princípios são baseados nesses pilares. Eles ajudam na implementação de sistemas mais robustos e seguros.

Confidencialidade: Por confidencialidade entende-se que os dados podem ser acessados apenas por usuários permitidos.

Integridade: Os dados podem apenas ser modificados ou deletados por pessoas autorizadas e de formas legais.

²⁴A notícia “Crimes cibernéticos custaram R\$ 15,3 bilhões no Brasil” pode ser encontrada no endereço: <http://convergenciadigital.uol.com.br/cgi/cgilua.exe/sys/start.htm?infoid=27750&sid=18>.

Disponibilidade: Garantia que o sistema e os dados estarão disponíveis para usuários autorizados, quando eles precisarem.

Em relação ao desenvolvimento seguro de aplicações, (HOWARD; LEBLANC, 2001) destaca alguns princípios que, de forma geral, estão relacionados com esse três pilares e que fornecem base para a construção de sistemas seguros. Estes princípios são descritos na sequência.

Aprender com os erros: Uma vez que um problema de segurança foi identificado, é importante enfrentá-lo como um oportunidade de aprendizado. Deve-se buscar as causas do erro, procurar outras áreas de código onde é possível que tenha ocorrido a mesma falha, avaliar como o erro poderia ser evitado e certificar-se que ele não acontecerá novamente.

Minimizar a área de ataque: Toda funcionalidade que é inserida em uma aplicação traz consigo uma certa quantidade de risco para aplicação em geral. O objetivo para o desenvolvimento seguro é reduzir o risco geral minimizando a área de ataque (CURPHEY *et al.*, 2005).

Um exemplo apresentado em (CURPHEY *et al.*, 2005) é uma aplicação *Web* que possui uma seção de ajuda e que possui a função de busca. A busca pode estar vulnerável a injeção de SQL²⁵. Se a funcionalidade de busca fosse limitada apenas a usuários autorizados, a possibilidade de ataque reduziria. Porém se a seção de ajuda fosse reformulada, mudando a interface para eliminar a função de busca, praticamente quase eliminaria a área de ataque desta seção, pois apenas tópicos poderiam ser exibidos em linguagem de formatação pura.

²⁵Injeção de SQL, do inglês *SQL injection*, é um tipo de ataque. Ele é abordado na Seção 4.2.

Dessa forma, a redução da área de ataque pode ser feita reformulando partes das aplicações *Web* para remover funcionalidades não muito usadas ou funcionalidades que são passíveis de serem removidas mantendo a usabilidade. Outra forma de reduzir a área de ataque seria manter apenas as principais funcionalidades ativas e então fornecer meios para que os usuários ativem as funções que por padrão vêm desativadas.

Utilizar a defesa em profundidade: A defesa em profundidade é um conceito bem simples de ser entendido. Para melhor entender considere os servidores de uma determinada empresa que hospeda algum tipo de aplicação *Web*. Essa empresa possui algum tipo de *firewall* para incrementar a segurança dos servidores. A defesa em profundidade, neste caso, instrui o desenvolvedor a considerar que o *firewall* foi comprometido e que não mais as aplicações podem contar com a proteção que era fornecida por ele, ou seja, a partir deste momento a aplicação deve ser capaz de garantir sua própria segurança.

Não deve-se deixar de implementar um mecanismo de segurança com a desculpa que existe outro mecanismo capaz de fornecer a segurança que é necessária. Se este mecanismo falhar a aplicação estará vulnerável, pois nenhum método de segurança foi implementado nela. A essência da defesa em profundidade é que em alguma etapa a aplicação precisará defender-se sozinha. Isto implica que é tarefa do desenvolvedor implementar métodos para prover segurança dela (HOWARD; LEBLANC, 2001).

Utilizar o menor privilégio: O princípio do menor privilégio recomenda que os usuários e processos devem ter o menor privilégio possível para realizar suas tarefas. No caso de alguma conta de usuário for comprometida ou seja possível injetar código em algum processo, o risco para a aplicação e seus dados é diminuído se a conta do usuário ou o processo tenham o

menor privilégio possível. Ao contrário, a ação maliciosa pode ser executada com privilégios elevados, o que irá comprometer a aplicação de forma mais crítica (HOWARD; LEBLANC, 2001).

Empregar padrões seguros: É desejável que a instalação padrão de um aplicativo seja segura “de fábrica”. Empregar padrões seguros significa fazer uso de métodos que aprimoram a segurança do aplicativo em sua instalação padrão, ou seja, aquela em que o usuário não se atenta em ativar/desativar mecanismos de segurança. Métodos como empregar análise de complexidade de senha no cadastro de usuários e usar um tempo para expiração de senha ajudam a melhorar a segurança. Aconselha-se que esses recursos venham ativados por padrão, porém é preciso que exista uma forma fácil de desativá-los. Nota-se que a partir deste ponto a responsabilidade passa a ser do usuário (CURPHEY *et al.*, 2005; HOWARD; LEBLANC, 2001).

Assumir que sistemas externos são inseguros: A suposição que sistemas externos são inseguros está ligada à defesa em profundidade. Muitas instituições e empresas usam processamento de dados de terceiros. Deve-se ter em mente que a política de segurança varia de organização para organização. Se a aplicação depende de dados de outros sistemas, é desejável considerar que esses dados são um possível ponto de ataque a ela, pois o desenvolvedor não tem controle do sistema externo e não tem conhecimento de como os dados são tratados ou sequer se são validados (CURPHEY *et al.*, 2005; HOWARD; LEBLANC, 2001).

Falhar de modo seguro: Ao falhar, a aplicação pode escapar de dois modos diferentes: o modo seguro e o modo inseguro. O modo seguro é aquele em que a aplicação não coloca em risco nenhum dado, ou seja, não expõe e

não permite a alteração dos dados. Outro ponto a ser levado em conta são as mensagens emitidas após a falha. É uma boa prática não emitir grandes quantidades de informações ao usuário, apenas informações suficientes para que seja possível notar que a aplicação falhou e a requisição não finalizou corretamente. Informações mais detalhadas sobre a falha podem ser registradas em arquivos de *logs*. O modo inseguro é o oposto. Dados são colocados em risco e diversas informações são exibidas ao falhar. Caso um invasor descubra que determinada aplicação falha de modo inseguro, ele poderá ter informações que o auxiliarão em um ataque maior (HOWARD; LEBLANC, 2001).

Lembrar-se que recursos de segurança é diferente de recursos seguros: Como pode ser visto em (HOWARD; LEBLANC, 2001), saber como os recursos de segurança funcionam é algo interessante, porém o mais importante é saber como usar tais recursos para reduzir ameaças à segurança. Um exemplo citado por Howard e Leblanc (2001) para relacionar recursos de segurança e recursos seguros é de uma entrevista de emprego para uma vaga de segurança. O entrevistador questionou o candidato sobre como usar um certo tipo de algoritmo de criptografia para evitar um problema. O candidato começou a explicar o funcionamento do algoritmo, o que é algo interessante a se conhecer. O problema foi que ele não sabia como aplicar tal recurso de segurança para reduzir ameaças ao sistema computacional em questão. Logo, recursos de segurança (o algoritmo de criptografia e seu funcionamento) é diferente de recursos seguros (aplicação do algoritmo para proteger recursos computacionais).

Nunca basear a segurança apenas na obscuridade: A segurança por obscuridade é um método ineficaz de segurança computacional que baseia-se em manter

em segredo os detalhes de como o sistema funciona (protocolos, algoritmos, sistemas internos, etc). Segurança por obscuridade baseia-se na falsa suposição que ninguém fora de um grupo confiável de pessoas será capaz de contornar o sistema. (MITNICK; SIMON, 2001).

Usar a segurança por obscuridade como parte da estratégia da defesa em profundidade é válido. De acordo com (HOWARD; LEBLANC, 2001), deve-se supor que o invasor tenha as mesmas informações (códigos e projetos) que a equipe de desenvolvimento.

Corrigir os problemas de segurança corretamente: Uma vez que um problema de segurança foi identificado, é importante desenvolver um teste para aquele problema. O objetivo deste teste é buscar em outras áreas do sistema este mesmo problema. É possível que o desenvolvedor tenha cometido o mesmo erro em diferentes partes. Tomando esta atitude, o problema será corrigido de forma correta, não dando brechas dele ocorrer mais vezes no sistema (CURPHEY *et al.*, 2005; HOWARD; LEBLANC, 2001).

Tais princípios são relativamente simples de serem implementados e trazem grandes benefícios ao sistema em questão. De fato, empregar padrões seguros ajudam a diminuir os riscos da aplicação. E aprender com os erros é sinal de maturidade no processo de desenvolvimento e isto irá garantir que sistemas futuros sejam imunes a determinados problemas.

3.4 Ameaças à segurança

Como visto em (NEWMAN, 2009), é comum ouvirmos em assuntos relacionados à segurança computacional algumas palavras-chave, tais como: *vulnera-*

bilidades, ameaças e medidas defensivas. A vulnerabilidade é um ponto onde o sistema computacional é suscetível a ataques. Já a ameaça é uma violação de segurança em potencial. Esta violação em potencial pode levar a exploração de uma vulnerabilidade. E finalmente as medidas defensivas são as ações tomadas para conter as vulnerabilidades e ameaças contra sistemas computacionais e informações.

Pode-se dizer que todos sistemas computacionais conectados à Internet estão sujeitos a ataques. O ataque, na maioria das vezes, é direcionado ao ponto onde o sistema é mais frágil, ou seja, aquele ponto que não conta com mecanismos de segurança adequados ou possui alguma falha. Diz-se então que este ponto é vulnerável. De acordo com Uchôa, Caminhas e Santos (2009), *vulnerabilidade* pode ser definida como qualquer fraqueza ou falha em um sistema que permite a um invasor violar sua integridade. O termo também pode ser aplicado para deficiências do sistema que impeçam sua rápida recuperação em caso de problemas nos equipamentos computacionais.

Na sequência são listados os pontos típicos de vulnerabilidades em sistemas computacionais.

Vulnerabilidades físicas: Este tipo de vulnerabilidade refere-se ao prédio da empresa e às salas de equipamentos. Se estes locais são inseguros, passíveis de invasão, os invasores entram e então podem vandalizar os equipamentos, roubar dispositivos de *backup* e documentos impressos. Este tipo de vulnerabilidade não ocorre apenas invadindo o local. Ele é também combinado com engenharia social²⁶ para facilitar a entrada e então dificultar a

²⁶Segundo (MITNICK; SIMON, 2001), engenharia social usa a influência e a persuasão para enganar pessoas convencendo-as que a pessoa é algo que na verdade ela não é. Como resultado, esta pessoa é capaz de pegar informações de outras com ou sem o uso de tecnologias.

descoberta de tal invasão. Normalmente funcionários de empresas confiam em visitantes por que eles estão usando uniformes de vendedores ou dizem que estão lá para consertar a máquina fotocopidora²⁷. Dispositivos de biometria, seguranças e cadeados juntamente com treinamento dos funcionários proporcionam uma importante defesa contra arrombamentos e invasões com engenharia social.

Vulnerabilidades naturais: Outro tipo de vulnerabilidade que afetam sistemas computacionais são as naturais. Como visto em (LEHTINEN; RUSSELL; GANGEMI, 2006), estes sistemas estão sujeitos a desastres naturais e ameaças do ambiente em que se encontram. Desastres tais como incêndio, inundação, terremotos, raios e perda de energia podem destruir os sistemas e seus dados. Poeira, humidade e variações de temperatura também podem afetar tais sistemas. No ambiente dos equipamentos é importante disponibilizar filtros e condicionadores de ar, bem como sistemas de energia redundantes para ajudar a garantir a integridade dos dados e dos equipamentos.

Vulnerabilidades de *hardware*: Certos tipos de falhas de *hardware* podem comprometer a segurança de um sistema computacional inteiro. Vulnerabilidades na BIOS (sistema básico de entrada e saída), em placas de rede e processadores podem causar um grande dano, pois neste tipo de vulnerabilidade a solução, geralmente, é trocar o *hardware* afetado. Um fato que envolveu vulnerabilidades de *hardware* foi com o vírus Stuxnet que, acredita-se, afetar apenas computadores com uma placa de rede específica²⁸.

²⁷Jim Stickley, especialista em segurança online e engenharia social, escreveu em outubro de 2011 uma de suas experiências. Ela pode ser encontrada em <http://computerworld.uol.com.br/seguranca/2011/10/31/seguranca-espionar-empresas-e-mais-simples-do-que-parece/>

²⁸Mais sobre o vírus Stuxnet pode ser encontrado em <http://olhardigital.uol.com.br/produtos/seguranca/stuxnet-o-virus-mais-sofisticado-que-ja-existiu>

Vulnerabilidades humanas: O ser humano, aquele que usa ou administra um sistema computacional, representa a maior das vulnerabilidades, segundo (LEHTINEN; RUSSELL; GANGEMI, 2006). Infelizmente pessoas podem ser subornadas ou coagidas para deixar alguma porta destrancada, fornecer uma senha ou fazer qualquer outra coisa que ameace a segurança do sistema ou do prédio de uma empresa.

As ameaças podem levar a exploração de uma vulnerabilidade. A definição de ameaça, de acordo com (NEWMAN, 2009), é qualquer ocorrência em potencial, seja acidental ou mal intencionada, que pode gerar efeitos indesejáveis em recursos de um indivíduo ou de uma organização. Entre os recursos estão dispositivos de *hardware*, *software*, bancos de dados, arquivos ou a própria rede.

Assim para conter as vulnerabilidades e ameaças contra sistemas computacionais e informações, (STALLINGS, 2008) destaca o uso de mecanismos de segurança. Estes são quaisquer processos (ou um dispositivo incorporando determinado processo) projetado para detectar, impedir ou permitir a recuperação de um ataque à segurança. Alguns exemplos de mecanismos são algoritmos de criptografia, assinaturas digitais e protocolos de autenticação.

3.5 Comentários finais

Neste capítulo, foi possível notar a grande dependência que a sociedade possui em relação aos sistemas computacionais. Estão presentes em muitas áreas como indústria, medicina, entretenimento, entre outras. Também foram apresentadas as ameaças a estes sistemas, bem como alguns princípios de segurança que se seguidos ajudam a manter dados e sistemas confiáveis.

No próximo capítulo, o termo segurança computacional será direcionado às aplicações *Web*, sendo que as vulnerabilidades mais comuns serão apresentadas.

4 SEGURANÇA EM APLICAÇÕES WEB

4.1 Comentários iniciais

Com a popularização da Internet e de serviços através dela, o tema segurança em aplicações *Web* é algo interessante a se discutir. O ambiente é bom para diversos lados: empresas que aumentam seus lucros através do comércio *online*, usuários que confiam em aplicações *Web* para lidar com suas informações pessoais e criminosos que podem obter muito dinheiro roubando informações de pagamentos com cartões de crédito e comprometendo contas bancárias.

Como mostrado no Capítulo 2, as aplicações *Web* fornecem diversas facilidades. Entre elas, o fator disponibilidade é um dos que merecem destaque. Como as aplicações estão na *Web*, teoricamente, elas podem ser acessadas por qualquer indivíduo com conexão à Internet. Ainda no Capítulo 2 foram descritas algumas funções mais comuns às aplicações *Web* atualmente, como compras, acesso a bancos, buscas, entre outras. Diversos são os dados gerenciados pelas aplicações. Empresas que intermediam compras na Internet por exemplo, PagSeguro²⁹ e PayPal³⁰, possuem dados de diversas pessoas assim como informações de seus cartões de créditos. As implicações geradas por um vazamento de informações são enormes. Falhas podem comprometer um grande volume de dados afetando muitas pessoas e também a credibilidade da empresa que fornece o serviço.

Este capítulo tem como objetivo abordar os principais problemas relacionados à segurança computacional envolvendo aplicações *Web*.

²⁹<https://pagseguro.uol.com.br/>

³⁰<https://www.paypal.com/>

4.2 Vulnerabilidades em aplicações *Web*

Como toda nova tecnologia, as aplicações *Web* trouxeram consigo novos tipos de vulnerabilidades e ataques associados. O primeiro são pontos de fraqueza na aplicação que são suscetíveis a ataques. O segundo são técnicas usadas por atacantes com o objetivo de explorar vulnerabilidades na aplicação. De acordo com (STUTTARD; PINTO, 2007), o principal problema de segurança em aplicações *Web* está no fato dos usuários poderem enviar entradas completamente arbitrárias às aplicações. Assim o passo inicial de todas aplicações é assumir que todas entradas, sem exceções, são maliciosas. Então seguir determinados passos para garantir que a entrada está “limpa” a ponto de não permitir que atacantes comprometam a aplicação. Entradas maliciosas podem interferir na lógica e comportamento da aplicação para ganhar acesso privilegiado a seus dados e funcionalidades.

Ainda de acordo com (STUTTARD; PINTO, 2007), este problema principal de envio de entradas arbitrárias manifesta-se de diferentes formas. Usuários podem interferir em qualquer parte do dado transmitido entre o cliente e o servidor. Controles de segurança implementados no lado do cliente podem ser facilmente burlados, como exemplo a validação de formulários usando *JavaScript*. Outra forma de interferência refere-se ao envio de requisições às aplicações *Web*. Os usuários podem enviar requisições e parâmetros às aplicações em diferentes sequências. Afirmarões de como o usuário irá interagir com a aplicação podem ser violadas. E por fim, usuários não estão restritos apenas aos navegadores *Web* para usar as aplicações. Diferentes ferramentas podem ser usadas para ajudar na exploração de aplicações. Como exemplo pode-se citar ferramentas que são capazes de enviar requisições e parâmetros para procurar e explorar vulnerabilidades.

Este problema citado é apenas um dos que levam a exploração de aplicações *Web*. É necessário conhecer tipos de ataques e vulnerabilidades associadas às aplicações para auxiliar na preparação de um esquema de segurança para elas. Assim, com o objetivo de reunir dados de vulnerabilidades de *sites* e obter um melhor entendimento sobre o cenário de vulnerabilidades em aplicações *Web*, o WASC (*Web Application Security Consortium*) [Consórcio de Segurança em Aplicações *Web*], juntamente com o apoio de indústrias como *Positive Technologies*, *Whitehat Security*, *Cenzic*, *HP Application Security Center*, *Veracode*, *Blueinfy*, *DNS, Encryption Limited*, elaborou o “WASC Web Application Security Statistics Project 2008” (GORDEYCHIK *et al.*, 2008). Este projeto tem como objetivo verificar quais classes de ataques são mais comuns e também comparar metodologias para identificar ataques.

O projeto possui estatísticas de vulnerabilidades de aplicações *Web* que foram coletadas durante testes de invasão, auditoria em segurança e outras atividades feitas por organizações que são membros do WASC até 2008. Os dados são de 12186 *sites* totalizando 97554 vulnerabilidades detectadas. As vulnerabilidades e ataques mais comuns de acordo com o projeto são as seguintes:

Cross-site scripting: Também conhecido como XSS, é um tipo de vulnerabilidade que permite injeção de códigos no lado cliente (*client-side*) de aplicações *Web*. Este tipo de vulnerabilidade habilita os atacantes a ignorar os controles de acesso, obter privilégios de acesso, desfigurar³¹ páginas *Web* e obter dados sensíveis como *logins*, senhas ou números de cartões de créditos (BODMER, 2007).

³¹O termo em inglês vem do verbo *to deface*, aqui traduzido como “desfigurar”, que é a ação de invadir um site e alterar o conteúdos das páginas *Web*. Sites brasileiros que veiculam notícias sobre segurança usam o termo “pichação”.

SQL injection: SQL (*Structured Query Language*) [Linguagem de Consulta Estruturada] é uma linguagem que permite interagir com banco de dados. As interações mais comuns são inserir, consultar, deletar e editar informações em bancos de dados relacionais. É bastante comum aplicações *Web* fazerem uso de banco de dados para o armazenamento de informações. A vulnerabilidade associada ao SQL é chamada de *SQL injection*, em português “injeção de SQL”. Se a parte da aplicação que lida com instruções SQL não filtra as entradas do usuário, é possível a injeção de instruções, o que permite ao atacante alterar a lógica das instruções e manusear (inserir, deletar, editar, consultar) as informações que estão no banco de dados.

Proteção insuficiente na camada de transporte: Em redes de computadores, a função da camada de transporte é fornecer comunicação lógica entre processos de aplicação que são executados em hospedeiros diferentes (KUROSE; ROSS, 2006). A comunicação lógica significa que as aplicações, que são executadas no cliente e no servidor, possuem a “sensação” de estarem conectadas diretamente através de uma via direta. Porém a realidade é que cliente e servidor podem estar em lados opostos do planeta e ligados por diferentes roteadores e enlaces.

A segurança aplicada à camada de transporte é o uso de *Secure Sockets Layer (SSL)* [Camada Segura de Sockets] e também *Transport Layer Security (TLS)* [Segurança na Camada de Transporte]. Eles são usados para prover criptografia à camada de transporte. Se a camada de transporte não contar com tais métodos de segurança, é possível que o tráfego entre cliente e servidor seja exposto a terceiros. Assim os dados podem ser interceptados e alterados antes que cheguem ao destino final, que pode ser tanto o cliente quanto o servidor.

HTTP *response splitting*: O HTTP *Response Splitting* é na verdade um vetor de ataques, pois ele é uma maneira, um método que leva a um objetivo final. Ele é a porta de entrada para a execução de outros ataques como o já abordado *Cross-Site Scripting*.

Cada linha em uma requisição HTTP é finalizada adicionando os caracteres `\r\n`. Assim é possível criar falsas requisições adicionando tais caracteres em uma requisição ao servidor. Se a entrada fornecida pelo usuário não for filtrada devidamente é possível realizar o ataque HTTP *Response Splitting*, onde os atacantes conseguirão manusear as requisições HTTP e consequentemente passar dados maliciosos às aplicações *Web*. Basicamente, usando os caracteres de fim de linha e de retorno é possível, em uma aplicação vulnerável, dividir a resposta HTTP do servidor em duas respostas HTTP, sendo a segunda de total controle do atacante.

Vazamento de informações: Como visto em (AUGER; BARNETT, 2010), o vazamento de informações, *information leakage* em inglês, é uma vulnerabilidade em que a aplicação revela dados sensíveis, como detalhes técnicos do servidor, da aplicação ou dados do usuário. Essas informações vazadas podem ajudar os atacantes a descobrir a quais tipos de vulnerabilidades a aplicação é suscetível.

Estas e outras vulnerabilidades e ataques serão abordadas com mais detalhes em capítulo posterior.

4.3 Comentários finais

Levando-se em conta as vulnerabilidades e ataques apresentados, bem como o uso de aplicações *Web* para lidar com operações sensíveis, nota-se a importância de metodologias para o desenvolvimento seguro, testes e avaliações de aplicações. Este tem sido o esforço da *OWASP (Open Web Application Security Project)*, que é o assunto abordado no capítulo seguinte.

5 OWASP - OPEN WEB APPLICATION SECURITY PROJECT

5.1 Comentários iniciais

OWASP (Open Web Application Security Project) é uma comunidade aberta dedicada a encontrar e combater as causas da insegurança em aplicativos. Todas as ferramentas, documentos e fóruns são gratuitos e abertos a qualquer um que esteja interessado em aumentar a segurança de aplicações (MEUCCI, 2008).

A *OWASP* não é afiliada com nenhuma empresa de tecnologia. Devido a liberdade em relação às pressões comerciais, as informações sobre segurança geradas pela comunidade são imparciais e práticas (MEUCCI, 2008).

A comunidade *OWASP* é auxiliada pela Fundação *OWASP*. Esta é uma entidade sem fins lucrativos que fornece a infra-estrutura, facilita o desenvolvimento de projetos e gerencia as conferências *OWASP Application Security*. Todos são bem-vindos a participar dos fóruns, projetos e conferências. A comunidade *OWASP* é o lugar ideal para aprender sobre segurança de aplicações e rede e também para construir uma reputação como *expert* em segurança (MEUCCI, 2008).

5.2 Projetos

Os projetos *OWASP* são divididos em duas categorias: desenvolvimento e documentação. Os projetos de documentação são:

Development Guide: É um guia que foca no desenvolvimento seguro de aplicações *Web*. Ele fornece descrições detalhadas de diversos aspectos pertinentes às aplicações *Web* (CURPHEY *et al.*, 2005).

Testing Guide: É um guia que aborda as seguintes questões: *o que testar, por que testar, quando testar, onde testar e como testar* aplicações *Web*. Em suma, o objetivo é a busca efetiva por vulnerabilidades em tais aplicações (MEUCCI, 2008).

Code Review Guide: É um guia que foca em mecanismos de revisão segura de códigos para determinadas vulnerabilidades. Inicialmente o assunto era abordado no *Testing Guide*, porém tornou-se muito extenso, então foi necessário um guia a parte para cobri-lo totalmente (STOCK *et al.*, 2008).

Top Ten: É um documento de alto-nível que foca nas vulnerabilidades mais críticas de aplicações *Web*. Tem como objetivo destacar as consequências das fraquezas de segurança mais comuns em aplicações *Web*, além de fornecer técnicas básicas para proteção contra tais vulnerabilidades (WILLIAMS; WICHERS, 2010).

Application Security FAQ: É um conjunto de perguntas e respostas mais frequentes sobre segurança de aplicações *Web* (PAKALA; NINER, 2011).

Legal: É um projeto que tem como objetivo garantir que a devida atenção à segurança está sendo dada nos diferentes estágios do ciclo de desenvolvimento de *software*. Ele também fornece um modelo de contrato de desenvolvimento de *software* seguro para auxiliar desenvolvedores e clientes a negociar termos e condições importantes relativas à segurança do *software* (WILLIAMS, 2011).

Metrics: É um projeto que pretende criar um conjunto inicial de métricas de segurança de aplicativos para a comunidade *OWASP* e, posteriormente, disponibilizar um fórum para a comunidade contribuir com novas métricas (BARTO, 2009).

Os principais projetos de desenvolvimento são *WebScarab* e *WebGoat*.

*WebScarab*³² é um *framework* usado para análise de aplicações que usam os protocolos de comunicação HTTP ou HTTPS. É escrito em Java, logo pode ser usado em diversas plataformas. Ele possui vários modos de operação. No modo mais comum, ele é usado para interceptar o *proxy*, o que permite que o usuário veja e modifique requisições criadas pelo navegador *Web* antes delas serem enviadas ao servidor. Também permite ver e modificar respostas do servidor antes de serem recebidas pelo navegador. A Figura 3 mostra as capturas feitas pelo *WebScarab* após alguma navegação no *browser*.

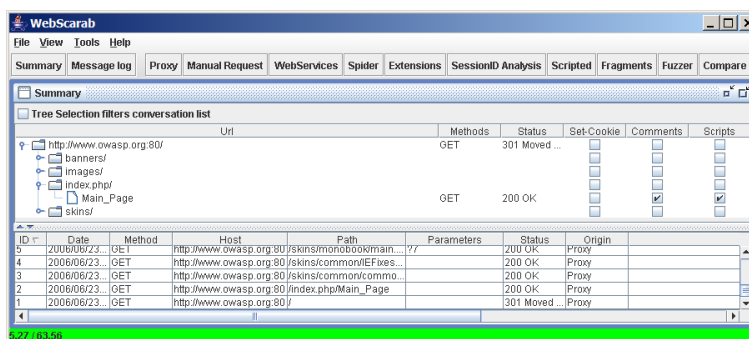


Figura 3: Captura de tela do *WebScarab* após alguma navegação no *browser*

*WebGoat*³³ é uma aplicação J2EE intencionalmente insegura que é mantido pela *OWASP*. É um ambiente de ensino realístico, projetado para ensinar lições de segurança em aplicações *Web*. Em cada lição os usuários devem demonstrar sua compreensão de um problema de segurança explorando uma vulnerabilidade real no *WebGoat*. Em uma das lições, por exemplo, o usuário deve usar *SQL injection* para roubar números de cartões de crédito falsos.

³²https://www.owasp.org/index.php/OWASP_WebScarab_Project

³³https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

5.3 OWASP Testing Guide

O *OWASP Testing Guide* Versão 3 (MEUCCI, 2008), abordado anteriormente, é um guia que foca na busca efetiva por vulnerabilidades em aplicações *Web*. Tal guia captura o consenso dos principais especialistas em segurança e auxilia na execução de testes com rapidez, precisão e eficiência. Um dos objetivos do Projeto *OWASP Testing* é ajudar no entendimento das seguintes questões: *o que é testar, o que testar, por que testar, quando testar, onde testar e como testar* aplicações *Web* e não apenas prover uma lista de verificação de problemas que devem ser abordados. Algumas questões relacionadas são apresentadas na sequência:

O que é testar? No contexto do guia de testes da *OWASP*, teste é um processo de comparação entre o estado de um sistema/aplicação e um conjunto de critérios (MEUCCI, 2008).

Por que testar? Outro objetivo do guia é ajudar a entender no que compreende um programa de testes. Além disso auxiliar na identificação dos passos que devem ser seguidos para ser possível implementar e colocar em prática um programa de testes em suas próprias aplicações *Web* (MEUCCI, 2008).

Quando testar? Muitas pessoas não testam suas aplicações até que ele esteja pronta e em funcionamento. Uma das melhores práticas para prevenir-se de erros é aperfeiçoar o ciclo de desenvolvimento adicionando segurança em cada uma das etapas (MEUCCI, 2008).

O que testar? Pode ser útil pensar no desenvolvimento de *software* como uma combinação de pessoas, processos e tecnologias. Se estes são os fatores que criam *software*, então é certo que tais fatores devem ser testados (MEUCCI, 2008).

Segundo (MEUCCI, 2008), o problema de *software* inseguro é talvez o desafio técnico mais importante do nosso tempo. É desnecessário dizer que não é possível construir uma aplicação segura sem efetuar testes de segurança. Ainda assim, o teste de segurança sozinho não é uma boa referência para quantificar o quanto uma aplicação é segura. Em contrapartida, tais testes possuem a capacidade de convencer os céticos que em determinado ponto da aplicação existe um problema. Desta forma, os testes de segurança tem se mostrado um ingrediente fundamental para organizações que tem que confiar no *software* que usa e/ou desenvolve.

5.4 Comentários finais

Neste capítulo, a comunidade *OWASP* e seus projetos foram apresentados. O *OWASP Testing Guide* Versão 3 (MEUCCI, 2008) será tomado como referência para os testes de segurança executados neste trabalho. No próximo capítulo, a metodologia de testes de aplicações *Web* da *OWASP* será apresentada.

6 TESTES DE SEGURANÇA EM APLICAÇÕES WEB

6.1 Comentários iniciais

Este capítulo em toda sua extensão tem como base o guia de testes da *OWASP - OWASP Testing Guide Versão 3.0* (MEUCCI, 2008). Tem como objetivo apresentar brevemente a metodologia de testes de segurança em aplicações *Web* da *OWASP* e não em ser uma descrição total do guia.

6.2 Metodologia de testes

Como apresentado no guia de testes, a metodologia de testes de segurança da *OWASP* é baseada na abordagem caixa preta. De acordo com Mall (2009), nesta abordagem os testes são desenvolvidos sem conhecimento sobre o projeto e código da aplicação a ser testada. Os testes são divididos em dois modos.

Modo passivo: nesta etapa o testador tenta entender a lógica da aplicação. Ao final desta etapa o testador pode ter conhecido todos os pontos de acesso da aplicação, tais como: cabeçalhos *HTTP*, parâmetros, *cookies*, entre outros.

Modo ativo: nesta etapa, a metodologia de testes de invasão da *OWASP* começa a ser utilizada. Os testes são divididos em nove sub-categorias:

- teste de autenticação;
- teste de autorização;
- teste de lógica de negócios;
- teste de validação de dados;
- teste de negação de serviço;

- teste de gerenciamento de configuração;
- teste de gerenciamento de sessão;
- teste de *Ajax*;
- teste de *web service*;

O modo passivo, que é a coleta de informações, e todas as sub-categorias do modo ativo serão detalhas nas seções a seguir.

6.3 Coleta de informações

A primeira fase a ser realizada em um teste de segurança de uma aplicação *Web* é a coleta de informações. É preciso reunir tantas informações quanto possível sobre a aplicação que será avaliada. Diversas ferramentas podem ser usadas, tais como: mecanismos de busca, *scanners*, ferramentas para envio de requisições *HTTP*, entre outras. Nas próximas subseções serão apresentados algumas formas para coletar informações que auxiliarão no teste.

6.3.1 Identificação de *robots*

Os *robots*, também chamados de *spiders* e *crawlers*, são programas criados para visitar URLs de sites de forma automatizada. A partir de uma lista de URLs, varrem a página e buscam outros URLs, essas novas encontradas são adicionadas à lista para posteriormente também serem visitadas. São usados por buscadores *Web* para fazer a indexação das páginas para as buscas e por *spammers* para capturar endereços de e-mail automaticamente.

O comportamento dos *robots* é especificado pelo Protocolo de Exclusão de Robôs (*Robots Exclusion Protocol*) no arquivo `robots.txt`. Este arquivo auxilia o testador na identificação da estrutura de diretórios da aplicação a ser avaliada, e, caso exista, pode ser localizado no diretório *Web* raiz. Um exemplo é apresentado na sequência.

```
User-agent: *  
# Directories  
Disallow: /includes/  
Disallow: /misc/  
# Files  
Disallow: /CHANGELOG.txt  
Disallow: /cron.php
```

Uma descrição detalhada das diretivas que configuram a maneira de como as aplicações ou *sites* são varridos por um *robot* pode ser encontrada em: <http://www.robotstxt.org/>.

6.3.2 Descoberta em motores de busca

Considerando o mecanismo de busca do Google, assim que o *robot* termina de varrer o conteúdo de um site, um índice de páginas é compilado para ser usado no processo de buscas. Se nenhuma especificação de comportamento para os *robots* foi definida, é possível que toda aplicação esteja indexada. Isto habilita o testador a encontrar mais informações sobre tal aplicação, incluindo diretórios e páginas usados pelos administradores.

No mecanismo de busca do Google, a descoberta de conteúdos pode ser feita usando operadores avançados de pesquisa. Para obter resultados de apenas um domínio específico pode-se usar o operador `site` da seguinte forma:

```
site:www.seusite.com.br
```

Mais sobre os operadores avançados de busca do Google podem ser encontrados no endereço: <http://support.google.com/websearch/bin/answer.py?hl=en&answer=136861>.

6.3.3 Identificação de pontos de entrada

A identificação dos pontos de entrada ajuda a reconhecer prováveis pontos de fraqueza da aplicação. Uma maneira para identificá-los é observar como navegador *Web* se comunica com a aplicação. Enquanto se navega pela aplicação é preciso dar atenção a todas requisições HTTP e aos parâmetros que são enviados e recebidos.

Nas requisições HTTP pode-se usar tanto o método `GET` quanto o método `POST` para passar parâmetros à aplicação. A principal diferença entre esses dois métodos é que o método `GET` usa o URL para realizar a passagem de parâmetros. Já o método `POST` passa os parâmetros no corpo da requisição HTTP. Assim para identificar os parâmetros em requisições usando `POST` é preciso usar um *proxy*, tal como o aplicativo *WebScarab* apresentado no Seção 5.2.

Ainda em requisições usando `POST`, é importante dar atenção aos campos escondidos de formulários. Esses campos não são vistos na página *Web*, para vê-los é preciso analisar o código-fonte HTML. Frequentemente tais campos possuem informações sensíveis, como quantidade de itens no carrinho de compras, valor da compra, entre outros. Usando um *proxy* é possível analisar todos os parâmetros

passados à aplicação, incluindo os campos escondidos, e também alterá-los antes de serem enviados. O código fonte de um campo escondido é apresentado na sequência:

```
<input type="hidden" name="parametro_nome" value="valor_do_parametro" />
```

Em relação as requisições HTTP, é importante verificar quais requisições usam GET e quais usam POST, identificar os parâmetros em ambas requisições, não deixando de lado os parâmetros passados usando os campos escondidos nas requisições POST. E em relação as respostas HTTP, é importante verificar onde *cookies* são criados e modificados e onde existem quaisquer tipos de redirecionamentos, em especial *403 Forbidden* e *500 Internal Server Errors*.

6.3.4 Avaliação da assinatura do servidor *Web*

Conhecer o tipo e a versão do servidor *Web* na qual a aplicação esta sendo executada ajuda significativamente, pois capacita o testador a identificar vulnerabilidades já conhecidas e escolher os métodos adequados para ser usados durante os testes.

A avaliação da assinatura do servidor *Web* pode ser feita automaticamente por ferramentas de segurança. O aplicativo *httprint* é uma ferramenta indicada para este tipo de avaliação. Ele possui um banco de assinaturas e a partir dele reconhece o tipo e versão do servidor em uso. Mais informações sobre o *httprint* podem ser encontradas na página <http://net-square.com/httprint/index.shtml>.

6.3.5 Análise de códigos de erros

Em testes de segurança é comum deparar-se com diversas mensagens de erros que são geradas pelas aplicações *Web* ou pelos servidores *Web*. Tais erros são de grande serventia aos testadores pois podem revelar informações sobre bases de dados, *bugs* e outros componentes diretamente ligados às aplicações e servidores.

Um exemplo prático de informações que são descobertas através de mensagens de erros é apresentado na sequência:

```
telnet www.ufla.br 80

Trying 200.131.250.54...
Connected to site.ufla.br.
Escape character is '^]'.
GET /pagina_errada.php HTTP/1.1

HTTP/1.1 400 Bad Request
Date: Thu, 15 Mar 2012 16:45:15 GMT
Server: Apache/2.2.16 (Fedora)
Content-Length: 304
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
...
<address>Apache/2.2.16 (Fedora) Server at site.ufla.br Port 80</address>
...
```

Tanto no campo `Server` da requisição quanto na página HTML retornada são exibidas informações sobre o servidor *Web* e também sobre o sistema operacional sobre qual o servidor *Web* esta sendo executado.

6.4 Teste de autenticação

Autenticação é o ato de estabelecer ou confirmar algo, ou alguém, como autêntico. Em segurança computacional, autenticação é o processo de tentar verificar a identidade digital do remetente de uma comunicação. Testar esquemas de autenticação implica em conhecer como funciona o processo de autenticação e usar tal conhecimento para burlar o mecanismo.

Em aplicações *Web*, o mais comum é o uso de formulários HTML para realizar a autenticação. Com seu uso é possível implementar um esquema mais sofisticado de autenticação em relação aos esquemas de autenticação baseados no protocolo HTTP (*Basic e Digest Access Authentication*). Um formulário simples de autenticação é apresentado na sequência.

```
<form method="POST" action="login.php">
  <input type="text" name="username">
  <input type="password" name="password">
</form>
```

Os próximos capítulos abordarão os testes para avaliar a segurança do processo de autenticação de aplicações *Web*.

6.4.1 Transporte de credenciais através de um canal criptografado

Testar o transporte das credenciais de usuários significa verificar como os dados de autenticação são enviados para o servidor da aplicação. Usar uma conexão criptografada evita que os dados sejam interceptados por usuários maliciosos.

O foco deste teste é avaliar se a aplicação *Web* toma as medidas necessárias para evitar interceptações de dados. O testador irá então verificar se os dados inseridos e enviados em formulários *Web* são transmitidos usando protocolos seguros, tal como HTTPS.

A verificação é feita usando um *web proxy* para analisar as mensagens de requisição da aplicação *Web*. Se o campo do URL da linha de requisição, que é a primeira linha de uma mensagem de requisição HTTP, possuir um URL especificando o protocolo HTTP então pode-se entender que os dados serão enviados sem criptografia. Um exemplo mostrando uma linha de requisição usando HTTP é apresentado na sequência:

```
POST http://www.example.com/AuthenticationServlet HTTP/1.1
```

Para um transporte seguro de dados deve-se usar o protocolo HTTPS. A linha de requisição ao usar este protocolo é mostrada a seguir:

```
POST https://www.example.com:443/cgi-bin/login.cgi HTTP/1.1
```

É possível verificar que o campo do URL especifica o uso do protocolo HTTPS, que usa vias criptografadas para o tráfego de informações, e impossibilita a leitura dos dados por terceiros.

6.4.2 Avaliação da enumeração de usuários

Existem aplicações *Web* que, por algum erro de configuração ou por alguma decisão de projeto, revelam quando um *login* existe ou não na aplicação. Desta forma, o objetivo deste teste é verificar se ao interagir com o mecanismo de autenticação é possível obter um conjunto de usuários válidos. Este conjunto pode ser usado em um ataque de força bruta, assunto abordado em capítulo posterior.

O testador irá enviar diferentes combinações de usuários e senhas e então observar a resposta da aplicação. A Tabela 1 mostra as situações possíveis para esta interação.

Tabela 1: Situações possíveis para enumeração de usuários.

	Usuário	Senha	Resposta da aplicação
Situação 1	Correto	Incorreto	A senha informada não está correta.
Situação 2	Incorreto	Incorreto	Usuário não reconhecido.

A **Situação 1** deixa a entender que o usuário informado está correto. Assim esta aplicação é vulnerável a enumeração de usuários, logo pode-se enviar entradas diferentes para o usuário e observar a resposta da aplicação. De acordo com esta situação é possível criar um conjunto de usuários válidos. Já a **Situação 2** mostra que o usuário informado não é reconhecido na aplicação.

Informações que podem ajudar na enumeração de usuários podem ser vistas também nos títulos das páginas, em páginas de recuperação de senha e também em códigos de erros do protocolo HTTP.

É importante saber que dependendo da política da aplicação, ao realizar diversas tentativas de “adivinhação” de usuários e senhas, pode ocorrer de contas serem bloqueadas ou ter o IP bloqueado no *firewall* do servidor.

6.4.3 Conta padrão de usuário

A maioria das aplicações *Web* são hospedadas em infraestruturas populares. O conjunto de *software* que é usado para que uma aplicação *Web* funcione é bem conhecido, e precisam ser configurados e customizados de acordo com a necessidade da aplicação. Também diversos dispositivos de *hardware*, roteadores de rede, servidores de banco de dados, entre outros, possuem interfaces *Web* para configuração e administração. O problema envolvendo as aplicações e conjunto de *hardware* citados refere-se às credenciais de autenticação que são fornecidas para uma autenticação inicial e de configuração. Frequentemente eles não são devidamente configurados e as combinações *login/senha* padrões são mantidas ativas em aplicações que estão em produção.

A causa deste problema pode ser identificada como:

- profissionais de TI³⁴ inexperientes que desconhecem a importância de alterar *login/senha* padrão;
- administradores e usuários de aplicações *Web* que usam senhas fracas;
- aplicações que vazam informações quanto a validade do *login* durante tentativas de autenticação;

O teste para este tipo de problema é feito tentando encontrar combinações usando credenciais padrões como: (1) *admin*; (2) *administrator*; (3) *root*; (4) *system*; (5) *guest*; (6) *operator*. Dependendo do tipo de aplicação que esta sendo testada, pode-se usar tais valores em português.

³⁴Tecnologia da Informação

Outra situação possível é relacionar o nome da empresa proprietária da aplicação *Web* com a credencial. E também verificar na página responsável pelo registro de novos usuários quais são as regras, tais como caracteres válidos, número máximo de caracteres, entre outros, para a criação de *login* e senha. Isto pode diminuir as possibilidades a serem testadas.

Alguns sites possuem um banco de senhas padrões e podem ser usados para auxiliar no processo de “adivinhação” de credenciais da aplicação que esta em teste. O site <http://www.virus.org/default-password/> possui senhas para diversos dispositivos de rede.

6.4.4 Teste de força bruta

A idéia de um ataque usando força bruta é enumerar todos os possíveis candidatos a solução de um problema e checar quando cada um deles o satisfaz. No contexto de aplicações *Web*, o ataque consiste em conseguir uma conta de usuário válida para acessar áreas restritas da aplicação. Se obtiver êxito, ou seja, se conseguir uma credencial válida, o atacante poderá acessar áreas que possuem dados pessoais dos usuários, tais como, documentos, dados bancários, entre outros. Também pode ser possível acessar a área gerencial da aplicação.

A grande maioria das aplicações têm optado por usar métodos de autenticação baseados em formulário HTML, pois dão maior liberdade para implementar mecanismos mais eficientes e seguros de autenticação a usar os métodos de autenticação providos pelo protocolo HTTP (*Basic e Digest*).

Diversas maneiras para realizar o ataque usando força bruta podem ser usadas, tais como: (1) ataque de dicionário; (2) ataque de busca; (3) ataque de busca baseado em regras. Os ataques de dicionários são feitos por ferramentas que usam

dicionários com palavras que possivelmente são usadas pelo usuário da conta. O ataque de busca é aquele que tenta cobrir todas as possibilidades possíveis com base em um conjunto de caracteres e tamanho de senha. Afim de reduzir o tempo de execução do ataque anterior pode-se usar regras para gerar os candidatos, caracterizando o ataque de busca baseado em regras.

O programa *THC Hydra*³⁵ pode ser usado para executar ataques de força bruta tanto em esquemas de autenticação usando formulários HTML quanto usando autenticação HTTP.

6.4.5 Contorno do esquema de autenticação

Aplicações *Web* podem possuir um esquema de autenticação que não forneça a segurança adequada, ou seja, que não seja capaz de garantir o devido controle para o acesso às áreas restritas. Isto pode dar brecha para um atacante ignorar o esquema de autenticação, habilitando-o a acessar páginas que deveriam ser acessadas somente após uma autenticação realizada com sucesso.

Algumas maneiras para ignorar o esquema de autenticação são apresentadas na sequência.

Requisição direta de página: caso a aplicação faça o controle de acesso apenas na página de *login*, o esquema de autenticação pode ser ignorado ao requisitar as páginas de forma direta. Pode-se acessar, por exemplo, a página `http://www.exemplo.com.br/painel.php`, pois a verificação de *login* foi feita, erroneamente, apenas para a página `http://www.exemplo.com.br/login.php`.

³⁵<http://www.thc.org/thc-hy>

Modificação de parâmetros: também é possível que uma determinada aplicação use apenas um *cookie* para controlar o acesso às áreas restritas. Tal *cookie* pode ser da seguinte forma: `autenticado=sim`. O testador pode mudar o valor do *cookie* e então será reconhecido como autenticado pela aplicação.

Desta forma, a função do testador é avaliar se existe alguma situação, na aplicação sendo testada, parecida com as situações descritas acima. Se sim, a aplicação é vulnerável e seu esquema de autenticação pode ser contornado.

6.4.6 Vulnerabilidades em *Esqueci minha senha* e *Lembrar senha*

No mecanismo de autenticação de aplicações *Web* é comum encontramos páginas específicas para recuperar ou redefinir senhas, e também a funcionalidade de *lembrar senha*. Esta provê comodidade aos usuários, pois permite acessos posteriores sem a necessidade de digitar a senha. Estes recursos podem inserir falhas no processo de autenticação.

Em relação a função *Esqueci minha senha*, na maioria das vezes o processo para recuperar ou redefinir senhas é feito enviando a nova senha ou *link* para redefinição de senha para o e-mail vinculado ao usuário. Percebe-se que isto implica que a aplicação confia totalmente na segurança do e-mail do usuário. Outra possibilidade é o uso de perguntas de segurança. Elas devem ser respondidas corretamente para prosseguir com a recuperação (redefinição) da senha.

Caso seja usado perguntas de segurança, pode-se então usar engenharia social para tentar obter a resposta da vítima. Para o processo de adivinhação da resposta é necessário ter em mente que a aplicação pode bloquear o mecanismo de recuperação após um determinado número de tentativas. Em relação a função *Lembrar*

senha, é necessário avaliar o uso de *cookies*, pois podem ser usados para armazenar as credenciais do usuário.

6.4.7 Avaliação do mecanismo de *logout*

A função de *logout* tem como objetivo encerrar todas as sessões do usuário com a aplicação e garantir que tais sessões serão destruídas e inutilizadas. Desta forma, este teste visa avaliar a possibilidade de reusar uma sessão após o processo de *logout*.

O processo de *logout* em aplicações *Web* ocorre após dois tipos de eventos: (1) o usuário encerra sua sessão (*logout*); (2) o usuário é automaticamente deslogado pela aplicação por inatividade (*timeout logout*). Se este processo não é executado de forma correta, um atacante pode ressuscitar uma sessão de um usuário real e então obter acesso à aplicação.

O teste mais simples a ser realizado no processo de *logout* é clicar no botão *Voltar* do navegador *Web* e então checar se o usuário ainda está logado na aplicação. Caso seja possível apenas visualizar páginas anteriores e não posteriores, isto é, novas páginas, significa que as páginas estão no cache do navegador.

Outra situação para teste é verificar o *cookie* após o *logout* e então restaurar o valor do *cookie* para seu estado inicial. Feito isto é necessário checar se o usuário ainda está logado na aplicação. Se sim, implica que a aplicação não faz controle de *cookies* ativos e inativos, e apenas as informações armazenadas no *cookie* são necessárias para garantir o acesso à aplicação.

Para o caso do usuário ser deslogado por inatividade (*timeout logout*), o teste é realizado checando quando o mecanismo existe e confirmando se todas as sessões

do usuário são apagadas ou inutilizadas após o *timeout*. Neste caso é preciso entender quando o *logout* é forçado pelo cliente ou pelo servidor. Se o *cookie* de sessão possui informações relacionadas ao tempo (horário de acesso, último acesso, etc), implica que o cliente esta envolvido com o processo de forçar o encerramento da sessão do usuário. Se este for o caso, é preciso alterar tais valores no *cookie* de sessão e avaliar se a sessão do usuário na aplicação foi prolongada.

Por fim, o fato de deslogar de uma aplicação não significa que o cache do navegador *Web* será apagado. É importante avaliar quando informações sensíveis são vazadas para o navegador.

6.4.8 Avaliação do CAPTCHA

CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*) [Teste de Turing público completamente automatizado para diferenciação entre computadores e humanos] é um teste usado para garantir que as ações em uma determinada aplicação não é gerada por um computador. A eficiência do CAPTCHA está em evitar ações automatizadas em páginas de *login*, registro de usuários, postagens em *blogs*, entre outros.

O teste para este tipo de mecanismo faz uso de um *proxy Web* para identificar todos os parâmetros que são enviados do cliente para o servidor juntamente com o CAPTCHA decodificado (valor digitado pelo usuário). Isto é feito pois podem existir parâmetros que possuem o valor do CAPTCHA decodificado. Outra possibilidade é tentar enviar um CAPTCHA decodificado anteriormente (já usado) com seu respectivo identificador. Caso for aceito, então é possível ignorar o mecanismo de CAPTCHA da aplicação.

6.5 Teste de autorização

Autorização é o processo que segue após uma tentativa de autenticação com sucesso. É nesta etapa que as permissões de acesso aos recursos da aplicação *Web* serão concedidas. Para este teste o testador precisará de credenciais válidas.

6.5.1 Teste de passagem de diretório

Servidores *Web* usam um diretório raiz para manter os arquivos e dados de aplicações *Web*. Este diretório é a base na hierarquia da aplicação *Web*. O teste de passagem de diretório (*directory traversal*), também conhecido como ataque *dot-dot-slash* (`../`) e como escalada de diretório, permite que atacantes visualizem diretórios e arquivos que supostamente não deviam visualizar, como exemplo o arquivo `/etc/passwd` em plataformas baseadas em Linux.

O teste é feito verificando partes da aplicação que aceitam conteúdos vindos do usuário e posteriormente verificando se tal conteúdo é validado de forma segura. Como exemplo o URL:

```
http://www.exemplo.com/perfil_usuario.php?item=pagina.html
```

A página `perfil_usuario.php` irá carregar informações vindas do arquivo `pagina.html`. Se a validação da variável `item` não for feita adequadamente é possível então carregar o arquivo citado anteriormente (`/etc/passwd`) usando a notação para retroceder um ou mais diretórios: `../../../../etc/passwd`.

Outra situação possível é incluir arquivos ou *scripts* vindos de outros sites, tal como:

<http://www.exemplo.com/index.php?arquivo=owasp.org/malicioso.txt>

No guia de testes da *OWASP* pode-se encontrar as diferentes codificações que representam a notação para retroceder diretórios, além de listar os diferentes caracteres separadores de diretórios nos diversos sistemas operacionais.

6.5.2 Burlagem do esquema de autorização

Este teste foca em avaliar como o esquema de autorização foi implementado para cada regra de acesso a funções e recursos da aplicação *Web*. A avaliação circunda a seguinte questão: *é possível acessar recursos e funções que supostamente era para ser acessadas apenas por um tipo de usuário?*

Usando a página `adicionaUsuario.php` como exemplo, suponha que ela faz parte das funções administrativas de uma aplicação *Web*. Ao executar a função para adicionar usuários, a seguinte requisição HTTP é gerada:

```
POST /admin/adicionaUsuario.php HTTP/1.1
Host: www.exemplo.com.br
[outros cabeçalhos HTTP]
userID=fakeuser&role=3&group=grp001
```

É preciso avaliar o que acontece se um usuário sem privilégios administrativos enviar tal requisição. O usuário será criado? Será possível logar como tal usuário? Se caso for possível, a aplicação é vulnerável e seu mecanismo de autorização é falho, ou seja, pode ser ignorado.

6.6 Avaliação da lógica de negócios

Testar falhas na lógica de negócios requer pensar de forma não convencional e tal teste não pode ser feito de forma automática por ferramentas de segurança. É preciso realizar uma análise profunda na aplicação e depende da habilidade e criatividade do testador. Cada caso é específico de cada aplicação.

Para ilustrar a situação de uma falha lógica, considere uma loja virtual onde o cliente adiciona créditos na conta e então pode comprar usando estes créditos. O processo feito pelo cliente para realizar uma compra seria:

1. adiciona créditos em sua conta;
2. busca produto ou lista diversos produtos;
3. adiciona produto no carrinho – nesta etapa a aplicação subtrai o valor do produto do total de créditos do cliente;
4. procede os passos de finalização da compra;

A lógica da aplicação seria basicamente a descrita acima. Uma falha (na lógica de negócios) seria o usuário informar um valor negativo para o número de itens e ter o valor do produto somado aos seus créditos.

Apesar de ser considerado uma arte, a descoberta de falhas lógicas, pode seguir os seguintes passos:

Conhecimento da aplicação: como as falhas são intrinsecamente ligadas a aplicação, ou seja, depende de como ela foi desenvolvida e como funciona, é

preciso possuir um grande entendimento sobre a aplicação. Tal conhecimento pode ser obtido através dos documentos que descrevem as funcionalidades, manuais, casos de uso, entre outros.

Coletagem dos dados: o testador precisa reunir informações sobre a aplicação, tais informações são extraídas nas seguintes situações: (1) cenários da aplicação; (2) fluxos de trabalho; (3) grupos de usuários e seus privilégios; (4) grupos de departamentos. Tais dados são essenciais para modelar os testes lógicos.

Modelagem dos testes lógicos: a partir dos dados coletados, o testador precisa modelar um teste para cada privilégio listado. E então verificar se alguma ação pode ser executada por um grupo que não possui privilégio para executá-la. Outra situação é navegar em ações que são projetadas passo a passo, focando em iniciar pelo meio do processo e observando o comportamento da aplicação.

Execução dos testes: nesta etapa é preciso observar as requisições HTTP, suas sequências, os campos escondidos de formulários, parâmetros, entre outros detalhes.

Este tipo de falha é muito perigosa, mais difícil de ser detectada, uma vez que não é possível realizar uma avaliação de forma automatizada. Outro ponto é que o teste para este tipo depende da experiência do testador e sua criatividade em manipular a aplicação.

6.7 Testando a validação de dados

Uma das premissas mais básicas em relação a segurança de aplicações em geral é: “Nunca confiar nas entradas do usuário”. A falta de validação dos dados vindos do usuário pode conduzir a diversas vulnerabilidades, como: *Cross Site Scripting*, *SQL Injection*, entre outras. Desta forma, este teste tem como objetivo avaliar se as entradas do usuário são suficientemente validadas antes de serem usadas.

6.7.1 Testando XSS não-persistente

O *Cross Site Scripting* (XSS), como já abordado anteriormente, é a possibilidade de manipular os parâmetros de entrada de uma aplicação *Web* para gerar uma saída maliciosa.

O XSS não-persistente, também conhecido como *Reflected Cross Site Scripting*, é aquele na qual a vítima carrega um URL ofensivo, ou seja, o ataque não é iniciado com o carregamento da aplicação vulnerável. Ele é iniciado com a ação de clicar em um *link* que possui parâmetros modificados e que levam a ações não desejadas. Este é o tipo mais frequente de ataque XSS. Os passos básicos para executar este ataque é modelar um URL malicioso e posteriormente convencer a vítima a acessá-lo.

Para o teste é necessário identificar os pontos de entrada e criar URLs maliciosas que apontam que tal ponto de entrada é vulnerável. Diferente do testador, que após identificar a vulnerabilidade irá reportar a falha, o atacante irá criar um URL que possa causar danos maiores à aplicação, seus dados e seus usuários.

Como exemplo considere um site que exibe uma mensagem de agradecimento e posteriormente um *link* para *download* de um arquivo. A partir do URL:

```
http://www.exemplo.com.br/download.php?user=Tulio
```

O site irá personalizar a mensagem de agradecimento. Se o site mostrar um *popup* após o carregamento do URL:

```
http://www.exemplo.com.br/download.php?user=<script>alert(12)</script>
```

Implica que tal site é vulnerável, pois foi possível inserir códigos *JavaScript* no parâmetro da aplicação. Se uma *popup* aparecer, ação feita com o trecho de código acima, significa que a entrada não foi validada e códigos maliciosos podem ser passados a aplicação.

As possibilidades a partir deste ponto são diversas. Pode-se alterar o *link* do arquivo que seria baixado pelo usuário, pode-se carregar um arquivo *JavaScript* que esta em outro servidor, entre outras. É importante notar que a validação da entrada deve ocorrer em diferentes codificações, pois pode-se filtrar `<script>`, mas pode-se não filtrar `%3cscript%3e`. Ambos representam a mesma entrada, porém em codificações diferentes.

A ferramenta *XSS-Proxy*³⁶ é uma poderosa ferramenta para ataques XSS e pode ser usada para auxiliar nos testes.

6.7.2 Testando XSS persistente

Aplicações *Web* que permitem que os usuários armazenem dados são potencialmente expostas ao XSS persistente. Este tipo ocorre quando a aplicação reúne

³⁶<http://xss-proxy.sourceforge.net/XSS-Proxy>

os dados do usuário e então os armazena para uso posterior. O fato é que se a entrada não for devidamente filtrada, os dados, que podem possuir ações maliciosas, aparecerão como parte da aplicação e conseqüentemente serão executados com os mesmos privilégios da aplicação. Assim sendo, este ataque é considerado bastante perigoso.

O teste consiste em identificar os pontos onde a entrada fornecida pelo usuário é armazenada e posteriormente exibida na aplicação. Tais pontos podem ser encontrados em páginas de “Perfil do Usuário”, que permite alterar nome, sobrenome, foto, etc. Lojas *online* que possuem carrinho de compras, os dados são armazenado e mostrados posteriormente.

Considere o seguinte trecho HTML:

```
<input type="text" name="email" value="aaa@aa.com" />
```

O testado então deve tentar injetar código fora da *tag* input. Um exemplo básico é tentar digitar o valor a seguir no campo input:

```
aaa@aa.com"><script>alert (document.cookie)</script>
```

Se a entrada não for filtrada, esse valor sera armazenado e toda vez que a página que exhibe o e-mail do usuário for carregada, o *script* será automaticamente carregado e exibirá um *popup* com o valor dos *cookies*. Caso ao submeter os dados e a palavra *script* for simplesmente apagada, isto indica que é possível ter algum tipo de tratamento para estas entradas maliciosas.

É recomendado que os testadores chequem o *RSnake*³⁷, este site disponibiliza uma lista extensa de ataques XSS e maneiras de burlar os filtros XSS.

³⁷*RSnake*: "XSS (Cross Site Scripting) Cheat Sheet- <http://ha.ckers.org/xss.html>

6.7.3 *SQL Injection*

O ataque *SQL Injection* consiste em manipular a entrada de dados da aplicação e enviar instruções SQL com o objetivo de invalidar a lógica feita pelo desenvolvedor. Um ataque com sucesso pode retornar ao atacante diversas informações sensíveis do banco de dados, além de poder permitir a manipulação dos dados, operações como ‘inserir’, ‘editar’ e ‘deletar’.

O primeiro passo do teste é verificar em quais partes a aplicação se conecta a um banco de dados. Páginas de autenticação, de busca, de listagem de produtos, entre outras, são fortes candidatas a usarem uma conexão com banco de dados. O testador então precisa listar todos os campos de entrada e posteriormente avaliá-los separadamente, pois um determinado campo pode estar protegido contra *SQL Injection* porém outro campo do mesmo formulário pode não estar protegido.

Os testes mais simples feitos nestes campos é usar aspas simples (') ou um ponto e vírgula (;). O primeiro é usado como terminador de *string* e o segundo como terminador de instrução. Ambos são usados para gerar erros. Uma mensagem de erro não tratada pode revelar informações importantes que auxiliam no prosseguimento do teste.

Considere a seguinte instrução:

```
SELECT * FROM Users WHERE Username=' $username' AND  
Password=' $password'
```

Esta consulta é usada para autenticar usuários. Caso ela retorne algum valor significa que as credenciais passadas existem e o acesso será liberado. Caso não retorne, significa que não existe e o acesso não será permitido.

Se o testador enviar tais valores:

```
$username = 1' or '1' = '1
```

```
$password = 1' or '1' = '1
```

E os dados usados na consulta não forem filtrados, a consulta resultante será:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND  
Password='1' OR '1' = '1'
```

Isto invalida a lógica que deveria ser usada para autenticar usuários e com esta entrada o acesso será permitido, mesmo a aplicação não conhecedo os verdadeiros valores de username e password.

Outra categoria de *SQL Injection* é chamada de *Blind SQL Injection*. Nesta variação o testador irá fazer consultas de inferências, tentando obter caracter por caracter um determinado campo. Com base no comportamento da aplicação ele poderá inferir se o caracter testado está ou não correto. Esta variação é usada no caso em que a entrada não é validada corretamente porém a mensagem de erro para a consultas com injeção de instruções SQL não revelam nada sobre a estrutura da consulta no banco de dados. É um tipo mais trabalhoso de teste e pode ser usado ferramentas para automatizar o processo. Mais sobre *Blind SQL Injection* pode ser encontrado em (SPETT, 2003).

É importante notar que os ataques do tipo *SQL Injection* podem variar de SGBD³⁸ para SGBD, assim o Guia de Testes da *OWASP* apresenta seções específicas para cada tipo de sistema.

³⁸Sistema Gestor de Base de Dados

6.7.4 XML Injection

Seja uma aplicação que usa XML para registrar usuários na aplicação. Isto é feito adicionando um novo nó `<user>` no arquivo XML exemplificado a seguir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid/>
    <mail>gandalf@middleearth.com</mail>
  </user>
</users>
```

Desta forma, o usuário irá preencher um formulário HTML, a aplicação receberá os dados digitados e então adicionará o novo nó.

O problema relacionado ao uso do XML esta na possível falta de validação dos dados que serão adicionados ao arquivo XML na estrutura especificada. Neste exemplo a estrutura usa as *tags* `<username>`, `<password>`, `<userid>` e `<mail>`. O usuário preencherá os campos *username*, *password* e *mail* e o campo *userid* será automaticamente inserido pela aplicação. Se não houver validação dos dados, o usuário pode ser capaz de comentar a *tag* `<userid>` adicionada automaticamente pela aplicação e então especificar um *userid* arbitrário. Como exemplo, veja as entradas abaixo:

```
Username: tony
Password: Un6R34kb!e</password><!--
E-mail: --><userid>0</userid><mail>s4tan@hell.com
```

A aplicação ao receber esses dados irá gerar a seguinte estrutura:

```
<user>
  <username>tony</username>
  <password>Un6R34kb!e</password><!--</password>
  <userid>500</userid>
  <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
</user>
```

Nesse caso, foi possível comentar as *tags* `<userid>`. O valor do `<userid>` foi alterado para 0, que pode ser o identificador do usuário administrador da aplicação. Com esta estrutura, o usuário `tony` ao logar-se terá privilégios de administrador.

O teste para checar tal vulnerabilidade consiste em inserir metacaracteres XML. Alguns exemplos de metacaracteres são: (1) aspa simples; (2) aspa duplas; (3) caracteres `< e >`; (4) *tags* de comentários (`<!-- e -->`); (5) caracter `&`; (6) *tags* CDATA. Estes metacaracteres podem ser usados para tentar invalidar a estrutura do arquivo XML e de acordo com as mensagens de erro, reunir algumas informações sobre o a estrutura do arquivo XML usado pela aplicação.

6.7.5 SSI Injection

Server-Side Includes (SSI) são diretivas que o servidor *Web* interpreta antes de enviar a página requisitada pelo usuário. As implementações mais comuns de *SSI* disponibilizam comandos para incluir arquivos externos, executar *scripts CGI*, executar comandos do sistema operacional, entre outros.

Na sequência são apresentados diretivas para incluir um arquivo externo e para executar um comando do SO:


```
<!--#include virtual="/footer.html" -->  
<!--#exec cmd="ls" -->
```

Como em toda situação de validação incorreta de entrada, um atacante pode fornecer uma entrada que, posteriormente, ao ser exibida será interpretado como uma diretiva *SSI*.

Com base no possíveis pontos de entrada vulneráveis, o testador deve checar se é possível usar os mesmos caracteres que são usados em diretivas *SSI*. Os caracteres estão listados a seguir:

```
< ! # = / . " - > and [a-zA-Z0-9]
```

Posteriormente, o testador deve analisar a página onde os dados fornecidos são exibidos. Nesta etapa será possível avaliar se a entrada foi devidamente validada ou se esta sendo interpretada como uma diretiva *SSI*.

6.7.6 HTTP Response Splitting

O ataque conhecido como *HTTP response splitting*, reúne a falta de validação da entrada do usuário mais o uso desta entrada para compor algum cabeçalho da resposta HTTP do servidor. A falta de validação permite que um atacante envie os caracteres CR e LF para a aplicação e, quando a aplicação fizer uso da entrada, tais caracteres serão interpretados como caracteres de separação de linhas da resposta HTTP.

Como exemplo, suponha uma aplicação que permita o usuário escolher entre interface simples e interface avançada. Após escolher, o usuário irá enviar a requisição para o servidor, informando que quer usar a interface avançada. Assim

que a aplicação recebe a escolha do usuário, ela irá responder com uma mensagem HTTP usando um cabeçalho que o redirecione para a interface avançada. A resposta poderia ser:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://aplicacao.com.br/index.php?interface=avancada
```

É possível notar que a aplicação faz uso da entrada do usuário para compor o cabeçalho `Location`, que realiza o redirecionamento. Se a entrada não for validada, o usuário pode inserir os caracteres CR e LF, dividindo a resposta do servidor em duas respostas HTTP. O usuário poderia então enviar o seguinte valor no parâmetro `interface`:

```
avancada%0d%0a<composição de outra mensagem HTTP>
```

Esta outra mensagem HTTP pode ser usada para montar outros ataques, tais como *Cross-Site Scripting (XSS)*, envenenamento de cache ou roubo de páginas com informações sensíveis. Mais detalhes sobre este ataque podem ser encontrados em (KLEIN, 2004).

6.8 Avaliação da negação de serviço

O ataque de negação de serviço (*DoS*) é usado para tornar um servidor inacessível. Em relação às aplicações *Web* pode ser possível tornar páginas e funcionalidades inacessíveis. Como exemplo, um usuário malicioso pode bloquear a conta de um usuário em uma aplicação após diversas tentativas de acesso.

Esta seção tem como objetivo abordar algumas formas de negação de serviço nas aplicações *Web*.

6.8.1 Avaliação do bloqueio de contas de usuários

Este teste envolve o sistema de autenticação de aplicações *Web*. Tem como objetivo checar se um atacante consegue bloquear uma conta de usuário válida enviando repetidamente senhas incorretas para a aplicação.

Este mecanismo de bloqueio de contas é uma maneira bastante utilizada para evitar descoberta de senhas usando ataques de força bruta. A ação tomada pela aplicação é bloquear a conta após um certo número de tentativas erradas. Desta forma, mesmo se o usuário legítimo da conta fornecer as credenciais corretas, será impossível autenticar-se até que a mesma seja desbloqueada. Assim, este mecanismo pode ser transformado em um ataque de negação de serviço.

O testador então deve checar se de fato a aplicação bloqueia contas após um número de tentativas erradas. Com um *login* válido é preciso enviar diversas senhas erradas para ser possível avaliar a situação.

Na Subseção 6.4.2, foi apresentado que aplicações *Web* podem ser vulneráveis a enumeração de usuários. Se um atacante obter um conjunto grande de usuário válidos, pode-se realizar o bloqueio de diversas contas automatizando a operação de envio de senhas erradas. Isto amplia os danos do ataque à aplicação *Web*.

6.8.2 Registro de dados do usuário no disco

O objetivo deste ataque é fazer com que a aplicação *Web* registre um *log* com grande quantidade de dados. Isto pode acontecer de duas maneiras:

- o testador envia uma requisição extremamente longa e a aplicação registra em *log* o valor diretamente sem checar seu tamanho;
- a aplicação valida os dados, porém registra-os de forma direta afim de manter os dados na forma na qual foi enviado para passar por um processo de auditoria ou similar;

Se uma determinada aplicação *Web* não aplica um limite máximo para cada entrada no *log*, então tal aplicação é vulnerável a este tipo de ataque.

A menos que o testador tenha acesso aos *logs* da aplicação e informações sobre os discos do servidor *Web*, é bastante difícil comprovar o sucesso ou não do ataque.

6.9 Teste de gerenciamento de configuração

Avaliar a infraestrutura na qual se encontra uma aplicação *Web* pode revelar muito sobre ela. Nesta seção, o foco dos testes é checar a configuração do transporte seguro, avaliar a infraestrutura da aplicação, avaliar o tratamento das extensões de arquivos, entre outros.

6.9.1 Testando SSL/TLS

O transporte seguro de dados é possível com o uso de SSL/TLS, o que resulta em tráfego HTTPS. Para garantir a segurança existe a necessidade de avaliar as configurações do SSL.

O primeiro passo do testador é obter as portas associadas a serviços SSL/TLS. A porta padrão é a 443, porém podem existir outras portas relacionadas. Esta

verificação pode ser feita usando *scanners* de vulnerabilidades, tais como *Nessus* e *NMAP*. O passo seguinte é avaliar as seguintes questões:

- a autoridade certificadora é confiável?
- o certificado é válido?
- o nome do site e o nome apresentado no certificado são os mesmos?

Geralmente os navegadores *Web* emitem alerta ao depararem-se com as situações descritas acima. A vantagem dos *scanners* é que pode ser usado *plugins* para checar tais situações e também detectar métodos fracos de criptografias, certificados vencidos ou que irão vencer nos próximos dias.

6.9.2 Avaliação da infraestrutura

Uma infraestrutura de servidores *Web* pode abrigar um grande número de aplicações *Web*. O gerenciamento de configuração é um passo fundamental no teste e implantação de aplicações. Se feito adequadamente ajuda a preservar a segurança das aplicações, pois se os elementos da infraestrutura não são devidamente configurados, novos riscos e novas vulnerabilidades podem surgir e comprometer as diversas aplicações.

A arquitetura da aplicação precisa ser revisada através de testes para determinar quais diferentes componentes são usados para implementar a aplicação *Web*. Pequenas configurações, como exemplo aplicações baseadas em *CGI*, podem usar um único servidor para executar o *software* do servidor *Web*. Configurações mais complexas, aplicações bancárias por exemplo, podem contar com diferentes servi-

dores, um para cada função, tais como servidor de aplicação, servidor de banco de dados, servidor *LDAP*, entre outros.

Para testar o gerenciamento de configuração da infraestrutura, o testador precisa avaliar os elementos da infraestrutura e as ferramentas administrativas buscando vulnerabilidades conhecidas e também manter um controle sobre as portas que são usadas pela aplicação.

A análise da infraestrutura é mais efetiva quando o testador possui informações internas do conjunto de aplicativos usados. Informações tais como versões usadas, correções aplicadas, entre outras. Desta forma é possível verificar diretamente com os fornecedores quais tipos de vulnerabilidades estão presentes em determinado elemento, sendo possível avaliar como tal falha pode afetar as aplicações da infraestrutura.

6.9.3 Testando o tratamento de extensões de arquivos

Determinar como o servidor *Web* lida com diferentes extensões de arquivos ajuda a descobrir quais tipos de arquivos serão executados e quais tipos serão retornados como texto puro. Um exemplo de extensão de arquivo que não é recomendado usar são aqueles com extensão `.inc`. Ao serem requisitados, o servidor irá retornar seu conteúdo como texto puro. O problema é que tais arquivos podem conter informações sensíveis sobre a aplicação.

O site <http://filext.com/> possui um banco de dados de extensões de arquivos. Ele pode ser usado para obter mais informações sobre os diferentes tipos de extensões.

6.9.4 Arquivos sem referência, antigos e de *backups*

Um valiosa fonte de vulnerabilidades encontra-se em arquivos antigos e de *backups*. Tais arquivos não possuem vínculo com a aplicação em produção e geralmente são esquecidos em diretórios que podem ser acessados pela *Web*. De forma geral, eles aparecem nas seguintes situações:

- após edições feitas diretamente no servidor, pois os editores costumam salvar arquivos de *backup*, arquivo `index.php~` no editor *Kate* por exemplo;
- após cópias manuais de segurança, arquivo `index.php.old` por exemplo;
- após *backups* compactados de toda a aplicação;
- ou por simplesmente deixar arquivos que não mais fazem parte da aplicação;

Todos eles podem conter informações internas sobre a aplicação, fornecendo credenciais e localizações de interfaces administrativas. No caso dos *backups* compactados, caso descoberto, todo código fonte da aplicação pode ser visualizado.

O teste para a descoberta de arquivos deste tipo envolve a análise do código fonte HTML e *JavaScript*. Ambos podem conter comentários sugerindo localizações de arquivos e funcionalidades escondidas na aplicação. O arquivo `robots.txt`, que provê instruções aos indexadores de conteúdo, também pode sugerir diretórios. Por fim pode-se tentar listar os diretórios do servidor *Web* diretamente pelo navegador, se o servidor não contar com uma configuração apropriada, será possível listar seus diretórios.

6.9.5 Testando os métodos HTTP

Os métodos do protocolo HTTP são usados para realizar ações no servidor *Web*. Os mais comuns são os métodos GET e POST, que são usados para acessar informações. Além destes métodos, existem outros tais como: (1) HEAD; (2) PUT; (3) DELETE; (4) TRACE; (5) OPTIONS; (6) CONNECT.

Recomendações de segurança apontam que os seguintes métodos devem ser desabilitados:

PUT: Permite que o usuário envie arquivos ao servidor. Pode ser explorado para enviar arquivos maliciosos.

DELETE: Permite que o usuário apague arquivos no servidor. Pode ser usado para desfigurar páginas.

CONNECT: Fornece a possibilidade para o usuário usar o servidor *Web* como um *proxy*.

TRACE: Retorna ao cliente qualquer conteúdo que foi enviado ao servidor. Pode ser usado para montar o ataque *Cross Site Tracing (XST)*. Mais detalhes em (GROSSMAN, 2003).

A maneira mais direta e efetiva para descobrir quais métodos HTTP estão habilitados no servidor *Web* é usar o método OPTIONS. Tal método retorna uma lista de métodos que são suportados pelo servidor. A partir desta lista o testador pode verificar a situação dos métodos citados acima.

6.10 Teste do gerenciamento de sessão

As sessões são usadas principalmente para manter um estado sobre as ações realizadas pelos usuários nas aplicações *Web*. Além disso elas aumentam a facilidade de uso da aplicação. Após autenticação, por exemplo, dados comprovando a autenticidade e validade do usuário são armazenados em sessões, tais como *cookies*, e são verificados a cada requisição HTTP durante o uso. Desta forma não existe a necessidade de autenticar-se a cada página requisitada.

Esta seção tem como objetivo testar o uso de sessões em aplicações *Web* e avaliar o quão seguro é o seu uso.

6.10.1 Avaliação do esquema de gerenciamento de sessões

As informações armazenadas nos *cookies*, usados para implementar as sessões, são de grande importância para o processo de segurança da aplicação. Se tais informações puderem ser alteradas, pode ser possível roubar sessões de usuário, ganhar privilégios de administração, entre outros. Assim este teste visa entender o mecanismo de gerenciamento de sessões e verificar se os *cookies* resistem as diferentes maneiras de burlagem. Os passos básicos para avaliar a segurança dos *cookies* são:

Coleta: é a obtenção de vários *cookies* para análise.

Engenharia reversa: consiste em entender como os valores dos *cookies* são gerados.

Manipulação: consiste em tornar um *cookie* válido na aplicação afim de ganhar privilégios.

Em relação ao uso de sessões, as seguintes características são desejáveis para manter o esquema seguro: (1) imprevisibilidade; (2) resistência a alterações; (3) validade; (4) atributo *secure*. O testador então avalia cada característica para verificar se o esquema provê a segurança necessária.

Testar a imprevisibilidade significa, avaliar vários *cookies* e tentar prever outros. Em outras palavras, buscar padrões para gerar outros *cookies*.

Testar a resistência a alterações significa avaliar se a aplicação executa uma checagem dupla no *cookie*. Como exemplo, além de enviar o valor do *cookie* tal como `admin=true`, enviar também o valor `true` na forma de *hash*, assim ao receber o valor, a aplicação verifica se o *hash* do valor atual do *cookie* combina com o *hash* original. Se não combinar significa que o valor foi alterado e ele deve ser descartado.

É desejável também que os *cookies* tenha uma validade adequada e posteriormente seja deletado quando não mais necessário.

Por fim, o atributo *secure* informa ao navegador que o *cookie* possui informações sensíveis e que não deve ser enviado em vias descriptografadas.

6.10.2 Avaliação dos atributos dos *cookies*

Devido a natureza sensível das informações carregadas pelos *cookies* é fundamental tomar as devidas ações para garantir sua segurança. Os *cookies* possuem diferentes atributos que podem ser usados para aumentar a segurança. Os seguintes podem ser usados: (1) *secure*; (2) *HttpOnly*; (3) *domain*; (4) *path*; (5) *expires*. Assim, o papel do testador é avaliar se tais atributos estão sendo usados corretamente pela aplicação.

Uma descrição dos atributos e como avaliá-los é apresentada na sequência.

Secure: Este atributo limita o uso dos *cookies* somente quando houver transmissão criptografada, HTTPS por exemplo.

HttpOnly: O uso deste atributo não é reconhecido em todos os navegadores, porém é desejável ser usado pois limita o uso dos *cookies* apenas ao protocolo HTTP, ou seja, impede que os *cookies* sejam manipulados por *scripts* no lado cliente, via *JavaScript* por exemplo. Desta forma o roubo de sessão feito com XSS é dificultado.

Domain: Este atributo limita o uso dos *cookies* somente ao domínio especificado. É preciso avaliar se o atributo *domain* não foi configurado vagamente. Como exemplo, se o *cookie* é necessário apenas no domínio `app.site.com`, o atributo deve ser setado especificamente a este domínio, privando outros como `outro_app.site.com` de recebê-lo.

Path: Tal como o atributo *domain*, este também deve ser setado de forma específica. Se a aplicação encontra-se em `app.site.com/versao2`, o atributo *path* deve ser setado como `path="/versao2/"`, com a barra no final para evitar que `versao2-vulneravel` também seja aceito.

Expires: Se o tempo para expiração do *cookie* for definido com datas futuras, e não com o fechamento do navegador, é preciso avaliar se tal *cookie* não carrega informações sensíveis.

A ferramenta *WebScarab* da OWASP e o *plugin FireCookie* do navegador *Web Firefox* podem ser usados para avaliar os *cookies* e seus atributos.

6.10.3 Avaliação de *session fixation*

Session fixation é um método de obter um identificador válido de sessão. Tal vulnerabilidade pode ocorrer em aplicações que não renovam a sessão após um processo de autenticação com sucesso. De forma geral a vulnerabilidade associada a *session fixation* funciona da seguinte forma:

- atacante cria um nova sessão na aplicação;
- o atacante faz com que a vítima acesse a aplicação usando o mesmo identificador de sessão criado por ele anteriormente;

Como já mencionado, se a aplicação não renova a sessão do usuário após a autenticação, tal aplicação pode ser vulnerável a *session fixation*. Usando o identificador de sessão criado pelo atacante, ele tenta convencer a vítima a acessar alguma página que altere a sessão do usuário para aquela sessão conhecida por ele. Se feito com sucesso, a sessão que é conhecida pelo atacante será associada ao usuário real da aplicação. Após autenticação na aplicação, os privilégios do usuário real serão associados a sessão criada pelo atacante, concedendo os privilégios também ao atacante, pois ele tem acesso ao *cookie*.

Mais detalhes sobre este vasto assunto pode ser encontrado em (SHIFLETT, 2004).

6.10.4 Avaliação CSRF

Cross Site Request Forgery (CSRF, pronuncia-se “*sea surf*”) é um tipo de ataque que aproveita a identidade e privilégios da vítima para executar ações não

desejadas na aplicação *Web* na qual ela esta autenticada. Isto é feito após a vítima carregar uma página que contém uma requisição maliciosa. No ponto de vista da aplicação a ação foi realizada pelo próprio usuário, no ponto de vista do atacante, o usuário executou a ação sem saber, pois desconhecia que tal página tinha uma ação maliciosa.

Como já abordado anteriormente, as aplicações *Web* geralmente fazem o uso de sessões para manter informações sobre o usuário e também informações sobre as credenciais de autenticação. Esses dados de sessão são automaticamente enviados ao servidor da aplicação, enquanto o usuário a usa, a cada requisição feita. Assim o usuário pode acessar diversos sites, quando a aplicação na qual ele está autenticado for usada, os dados de sessão, anteriormente definidos, serão também enviados.

O ataque CSRF inicia quando um atacante consegue obter *links* válidos da aplicação, ou seja, *links* que ao serem acessados realizam algum tipo de ação na aplicação. O atacante então convence a vítima a acessar tal *link* ou envia o *link* de uma página por ele trabalhada que executa a requisição maliciosa. Isto pode ser feito usando a *tag* `img` e colocando o *link* da ação maliciosa no atributo `src`, tal como:

```

```

Tal página preparada pelo atacante contém um *link* válido da aplicação *Web* e ele executa a ação de deletar usuários. A vítima ao acessar esta página irá carregar também o *link* `http://aplicacao.com/deletar_usuarios.php` executando a ação de deletar usuários. O fato é que o usuário deve estar logado na aplicação, pois ao executar tal *link* o navegador da vítima automaticamente irá enviar também os dados de sessão dela. Desta forma a ação será executada com sucesso aproveitando-se da identidade e privilégios do usuário na aplicação.

Entre as ações que podem ser tomadas para proteger uma aplicação de tal vulnerabilidade é o uso de informações de sessão a nível do URL. Isto tornará o trabalho do atacante mais complexo, pois com as informações de sessão será mais complicado para ele conhecer a estrutura dos *links* válidos na aplicação. Outra ação é o uso de janelas *popup* para a confirmação destas ações.

Para o teste, o testador deve possuir *links* válidos da área que necessita de autenticação para acesso. Se o testador possuir credenciais válidas, ele irá assumir o papel de atacante e vítima, ao mesmo tempo, para avaliar se a aplicação permite tal ataque.

Mais detalhes sobre o ataque CSRF podem ser encontrados em (WATKINS, 2001).

6.11 Avaliação da técnica *Ajax*

Como abordado no Seção 2.5, *Ajax* é uma técnica que agrupa diversas tecnologias para enviar requisições assíncronas ao servidor e trabalhar a resposta do mesmo. De forma geral, as vulnerabilidades usando *Ajax* são semelhantes as requisições comuns, sem uso de *Ajax*. Assim a maioria dos testes, extensivamente apresentados, são aplicáveis, pois o que muda é a forma da requisição de síncrona para assíncrona.

O teste consiste em identificar os destinos das requisições HTTP assíncronas e então analisá-los em busca de vulnerabilidades, tais como: alteração de parâmetros, *SQL Injection*, entre outras. A identificação dos destinos das requisições pode ser através da análise do código fonte *HTML* e *JavaScript*. Outra forma é usar um *proxy* para interceptá-las. Com os pontos de destinos o testador pode avaliar o formato das requisições.

O *plugin FireBug*³⁹ do navegador *Web Firefox* pode ser usado para testar o uso de *Ajax* pela aplicação *Web*. É possível facilmente identificar os destinos das requisições feitas através do objeto *XmlHttpRequest* e também identificar a estrutura dos dados enviados. Mais detalhes sobre o uso do navegador *Firefox* para explorar aplicações *Web* que usam a técnica *Ajax* podem ser encontrados em (SHAH, 2006). E maiores informações sobre os perigos que surgem com o uso de *Ajax* podem ser encontrados em (HOFFMAN, 2006).

6.12 Avaliação de *web services*

De acordo com (HAAS, 2004), *web service* é um sistema modelado para ser usado entre diferentes máquinas através da rede. A interface do serviço é descrita no arquivo WSDL (*Web Services Description Language*), informações de como acessá-lo, operações suportadas, formato das entradas, entre outras, são especificadas nele. A interação entre as máquinas é feita através de mensagens SOAP (*Simple Object Access Protocol*). Tal protocolo especifica como deve ser feita a troca de informações entre eles.

As vulnerabilidades em *web services* são similares as vulnerabilidades já descritas extensivamente neste capítulo, tais como *SQL Injection* e vazamento de informações, porém os *web services* podem também ter vulnerabilidades relacionadas ao *parser* das mensagens SOAP.

Esta seção abordará os testes que podem ser feitos em *web services*.

³⁹<https://addons.mozilla.org/pt-br/firefox/addon/firebug/>

6.12.1 Avaliação do WSDL

A partir do arquivo WSDL, aquele que descreve o *web service*, o testador pode identificar os métodos e tipos de dados esperado pelo *web service*. Desta forma, é necessário avaliar se é possível invocar operações não permitidas e se existem métodos que retornam informações sensíveis.

Um situação possível de ser encontrada é a existência de uma interface *Web* para invocar um *web service*. Tal interface pode mostrar apenas algumas operações, porém se o WSDL for checado, o testador pode encontrar outras operações não listadas na interface. Usando o *WebScarab* por exemplo, é possível alterar a requisição *SOAP* e invocar o método que a interface *Web* não listou.

6.12.2 Avaliação da estrutura do XML

A forma de comunicação entre o *web service* e o cliente é feita usando a estrutura de arquivos XML. Assim tais arquivos devem estar bem formados para que o *web service* funcione adequadamente, caso contrário o *parser XML* do *web service* pode falhar.

Os ataques envolvendo XML envolvem requisições *SOAP* com a estrutura errada e com tamanhos acima do normal. A função do testador então é interagir com o *web service* enviando requisições mal formadas e com tamanho grande e, conseqüentemente, observar o comportamento do *web service*.

6.12.3 Avaliação do conteúdo do XML

Através da requisição *SOAP*, o cliente realiza chamadas de funções e especifica parâmetros para a realização de alguma tarefa no *web service*. Este teste consiste em avaliar os parâmetros enviados. O mais comum é a injeção de códigos SQL.

Como exemplo, considere um *web service* que obrigue o usuário a autenticar-se. Se as credenciais (*login* e senha) enviadas pelo usuário não forem validadas, ele será capaz de usar o *web service* com privilégio de usuário autenticado. Isto pode ser possível ao enviar parâmetros que tornam a condição de autenticação sempre verdadeira, ou seja, fazendo uso de *SQL Injection*, como já abordado anteriormente.

6.13 Comentários finais

Neste capítulo, foi apresentado uma idéia dos testes que podem ser realizados em aplicações *Web*. Para uma versão mais aprofundada das abordagens apresentadas é aconselhável consultar *OWASP Testing Guide Versão 3* (MEUCCI, 2008). Vale notar que o capítulo em toda sua extensão tomou como base o guia acima citado.

No próximo capítulo, aplicações reais serão colocadas em prova usando a metodologia de testes de aplicações *Web* da *OWASP*.

7 ANÁLISE E DISCUSSÃO

7.1 Comentários iniciais

Com o objetivo de colocar em prática a metodologia de testes de segurança de aplicações *Web* da *OWASP*, avaliações foram realizadas em três aplicações de *e-commerce*, são elas: (1) *e-commerce real*; (2) *e-commerce PrestaShop*; e (3) *e-commerce básico* implementado segundo as orientações para desenvolvimento seguro da *OWASP*.

Nas próximas seções são apresentadas as análises de segurança para cada *e-commerce* citado.

7.2 *E-commerce real*

O *e-commerce real* é uma loja especializada em produtos relacionados à informática. Ela possui avaliação de *loja diamante* pela empresa e-bit Informação⁴⁰. A medalha diamante é concedida às lojas virtuais que são avaliadas positivamente nos quesitos como facilidade de comprar, preço, manuseio dos produtos, cumprimento do prazo de entrega, informações dos produtos, entre outros. O *e-commerce real* ainda possui um selo de *site blindado*. Tal selo pode ser visto na Figura 4.



Figura 4: *E-commerce real*: selo indicando site blindado

⁴⁰<http://www.ebit.com.br/>

A empresa *Site Blindado S.A*⁴¹ submete diversos sites a uma bateria de testes de vulnerabilidades, que segundo a empresa “*significa que o website está protegido contra tentativas de exploração e obtenção de informações confidenciais não autorizados através das técnicas de invasão mais conhecidas*”. Ainda segundo a empresa, se um site possui tal selo: “*Compre tranquilo - seus dados estão guardados com segurança*”.

Por motivos de ética, o nome da loja virtual, que é pioneira no comércio eletrônico brasileiro, não será divulgado neste trabalho e será tratada apenas como *e-commerce real*.

7.2.1 Teste do e-commerce real

A loja virtual possui a capacidade de trafegar os dados através de um canal criptografado, porém seu uso não é forçado automaticamente, o que significa que se o usuário acessar a aplicação usando `http://<loja>` todo o tráfego será feito em HTTP. Em contrapartida, se o usuário acessar usando `https://<loja>` o tráfego é feito de forma segura usando HTTPS. Como a aplicação não força o uso do HTTPS, pode acontecer do usuário usar o HTTP (simples) e ao realizar seu *login*, um atacante monitorar a rede e capturar tais credenciais. As informações sobre o certificado da loja podem ser vistas na Figura 5.

Em relação a enumeração de usuários, a loja não emite informações diferentes quando recebe credenciais inteiramente erradas ou quando recebe apenas a senha errada. A única mensagem exibida pode ser vista na Figura 6 que segue.

Na Figura 6 foi possível verificar que o mecanismo de autenticação não vaza informações que torne possível o agrupamento de usuários válidos da aplicação.

⁴¹<https://selo.siteblindado.com.br>

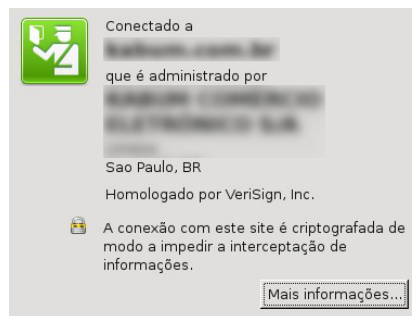


Figura 5: *E-commerce real*: informações de conexão segura

Atenção: E-mail ou senha inválidos.

Figura 6: *E-commerce real*: mensagem informando credenciais erradas

Porém foi possível verificar que o mecanismo de recuperação de senha e o cadastro de novos clientes especificam quando o e-mail existe ou não na loja virtual.

Para a recuperação de senha apenas o e-mail é requerido para iniciar o processo de recuperação. O mecanismo deste *e-commerce real* envia uma cópia das credenciais do usuário para o e-mail cadastrado. Ao inserir um e-mail válido, a seguinte mensagem é retornada pela aplicação:

Atenção: Uma cópia de sua senha foi enviada para seu e-mail.

Figura 7: *E-commerce real*: mensagem informando a ação do mecanismo de recuperação de senhas

A Figura 7 fornece detalhadamente o que foi feito pelo mecanismo de recuperação de senha. Isto implica que a aplicação confia totalmente no e-mail do usuário como alternativa para recuperar o acesso à loja virtual. Ainda no mecanismo de recuperação de senhas, ao inserir um e-mail inválido, a mensagem apresentada na Figura 8 é retornada pela aplicação.

Atenção: E-mail não está cadastrado no sistema.

Figura 8: *E-commerce real*: mensagem informando a inexistência do e-mail na loja

A partir da Figura 8 é possível realizar a enumeração de e-mails válidos na loja virtual. Se a mensagem informando que uma cópia da senha foi enviada para o e-mail for retornada, implica que o e-mail é válido e o atacante pode tentar obter acesso ao e-mail do usuário e conseqüentemente conseguirá acesso aos dados do usuário no *e-commerce real*. A enumeração de e-mails também pode ser feita ao cadastrar uma nova conta na loja virtual. Se um e-mail já existir, a mensagem da Figura 9 é retornada.

Atenção: Este e-mail já está cadastrado no site.

Figura 9: *E-commerce real*: mensagem informando a existência do e-mail na loja

A página responsável pela autenticação de usuários, faz controle das mensagens de erros através de uma variável. Desta forma pode-se alterar o valor de tal variável para analisar o erro retornado pelo mecanismo de autenticação. Tais mensagens são numeradas de um a seis, como segue.

1. `http://<loja>/login.cgi?msg=1`
2. `http://<loja>/login.cgi?msg=2`
3. `http://<loja>/login.cgi?msg=3`
4. `http://<loja>/login.cgi?msg=4`
5. `http://<loja>/login.cgi?msg=5`
6. `http://<loja>/login.cgi?msg=6`

As mensagens exibidas pelo *e-commerce real* até agora, são apresentadas nos URLs acima enumerados. Assim não seria necessário realizar as ações para verificar se a loja é ou não vulnerável, foi possível verificar todas as mensagens de erro usadas na página de *login*, simplesmente alterando o valor da variável `msg`.

O mecanismo de *login* cria um *cookie* com o nome `codigocrypt` após autenticação bem sucedida. Ele possui uma informação criptografada. Para testar como é feito o gerenciamento das sessões no *e-commerce real*, salvou-se o *cookie* e então verificou-se o comportamento da função *logout*. Foi possível notar que o *cookie* `codigocrypt` foi apagado. Após *logout*, o *cookie* foi restaurado e, sem informar nenhuma informação sobre as credenciais, o usuário voltava a ser considerado como autenticado na aplicação. A partir disso, pode-se consultar as diversas informações de cadastro do cliente, tais como CPF, RG, endereço completo e telefones. Se um atacante for capaz de obter o valor do *cookie* `codigocrypt` de algum usuário, o atacante será também capaz de acessar as informações cadastrais de tal usuário na loja virtual, pois tal loja não tem controle sobre a autenticidade do *cookie*. Assim verificou-se que a aplicação não tem controle sobre a validade de seus *cookies*.

De posse do valor do *cookie* `codigocrypt`, observou-se que seu valor era passado como parâmetro em diversas páginas para uma variável de nome `id_cliente`. Assim, salvou-se o valor do *cookie* e então deletou-se tal *cookie*. Foi possível acessar diversas páginas que possuíam a variável `id_cliente` como parâmetro. Concluiu-se que a aplicação garante acesso às páginas apenas com o valor do *cookie* `codigocrypt`, não sendo preciso tê-lo no cache do navegador *Web*. Algumas páginas que possuem tal parâmetro são apresentadas na sequência, todas elas exibem informações sensíveis.

```
http://<loja>/mc_cadastro.cgi?id_cliente=<valor cookie codigocrypt>  
http://<loja>/mc_login.cgi?id_cliente=<valor cookie codigocrypt>  
http://<loja>/mc_senha.cgi?id_cliente=<valor cookie codigocrypt>
```

Desta forma é possível ignorar o esquema de autenticação, pois apenas com a posse do valor do *cookie* *codigocrypt* é possível checar as informações cadastrais dos clientes, bem como a senha atual.

Todos os *links* acima citados, estão na página `http://<loja>/minha_conta/principal.cgi`. Foi possível verificar que mesmo após clicar em SAIR, a página não é redirecionada para outra e os *links* acima citados continuam válidos, ou seja, ainda é possível verificar as informações cadastrais e senha do usuário. Se um usuário clica em SAIR e deixa o navegador aberto, algum outro usuário pode clicar em tais *links* e acessar tais informações. Isto por que a aplicação não redireciona o usuário após o *logout* e a página já possui os *links* montados adequadamente fazendo uso do valor do *cookie* *codigocrypt*.

Concluindo o mecanismo de autenticação, verificou-se que não existe nenhum mecanismo de bloqueio de tentativas erradas de autenticação, isso significa que um ataque de força bruta pode ser executado sem problemas. A aplicação não bloqueia após certo número errado de tentativas. É importante notar que nada está sendo afirmado sobre o servidor que hospeda a loja virtual. Tal servidor pode ter um *firewall* que bloqueia várias tentativas seguidas de um mesmo endereço IP.

A página de cadastro de clientes não possui nenhuma validação dos dados requeridos. Não existe nenhuma restrição em relação a senha do usuário, que pode até mesmo ser de apenas um caracter.

A parte com a vulnerabilidade mais crítica do *e-commerce real* foi encontrada na geração de boletos das compras. Após realizar uma compra, diversas formas

de pagamento podem ser escolhidas. Independentemente do modo escolhido é possível gerar um boleto para pagamento. Desta forma, após uma compra real, um *link* para geração e posterior impressão do boleto foi gerado. O URL para gerar o boleto é apresentado na sequência.

`http://<loja>/boleto.cgi?numero=1266782&data=1319407776`

O boleto é gerado usando os parâmetros `numero` e `data`. O primeiro é o número do documento, ou seja, número do boleto. O segundo é o horário em que a compra foi feita. O modo usado foi o *UNIX TimeStamp*, que é um número de segundos desde 1/1/1970. Este valor pode ser facilmente convertido em *Sun, 23 Oct 2011 22:09:36 GMT* para descobrir a data representada por 1319407776 segundos.

Foi possível constatar que nenhuma informação de sessão é exigida para acessar o boleto. Para explorar tal situação, foi desenvolvido um *script* em PHP. O *script* foi usado para acessar o URL de geração de boletos com diferentes parâmetros.

A idéia do *script* foi manter o parâmetro `numero` e ir decrementando o parâmetro `data` em uma unidade. Acessava-se o URL e verificava-se se um boleto era gerado com sucesso. Se não fosse gerado, o parâmetro `data` era decrementado novamente até que um boleto fosse encontrado. Após se encontrado, o parâmetro `numero` era decrementado e acessava-se com a mesma `data`, seguindo o ciclo em decrementar o parâmetro `data`. Em 40 a 60 minutos de execução foi possível obter aproximadamente 80 boletos. Um atraso no acesso aos *links* foi colocado afim de evitar bloqueios.

O grande problema da página de geração de boletos é que ao descobrir a lógica usada para gerá-los é possível obter vários deles. Eles trazem informações como nome e endereço completo do cliente. Outro fator importante é que a loja virtual em questão oferece desconto de 15% nas compras com boletos, ou seja, para compras à vista o desconto é bastante atrativo, cativando possíveis clientes. A Figura 10 mostra um boleto do *e-commerce real* obtido através do *script*.

Itaú Banco Itaú S.A. 341-7		34191.76015 26670.500276 92646.750007 9 51340000056421			
Cedente		Agência/Código do Cedente	Espécie	Quantidade	Nosso número
Número do documento 01266705		CPF/CNPJ	Vencimento	Valor documento	
			28/10/2011	564,21	
(-) Desconto / Abatimentos	(-) Outras deduções	(+) Mora / Multa	(+) Outros acréscimos	(=) Valor cobrado	
Sacado					
Instruções		Autenticação mecânica			
Cliente Itaú: Pague também no BankFone, BankLine ou Caixas Eletrônicas		Corte na linha pontilhada			
Itaú Banco Itaú S.A. 341-7		34191.76015 26670.500276 92646.750007 9 51340000056421			
Local de pagamento ATÉ O VENCIMENTO, PODE SER PAGO EM QUALQUER BANCO OU VIA INTERNET		Vencimento		28/10/2011	
Cedente		Agência/Código cedente			
Data do documento	Ng documento	Espécie doc.	Aceite	Data processamento	Nosso número
21/05/2012	01266705	R\$	N		
Uso do banco	Carteira	Espécie	Quantidade	Valor	(=) Valor documento
176	R\$				564,21
Instruções (Texto de responsabilidade do cedente)				(-) Desconto / Abatimentos	
ATENÇÃO SR.(A) CLIENTE:				(-) Outras deduções	
O pedido só é enviado após o pagamento deste boleto				(+) Mora / Multa	
Não confunda pagamento com agendamento - Pague o boleto com a data do próprio dia do pagamento				(+) Outros acréscimos	
				(=) Valor cobrado	
Sacado		Cód. baixa			
Sacador/Avalista		Autenticação mecânica - Ficha de Compensação			
					
Corte na linha pontilhada					

Figura 10: *E-commerce real*: boleto obtido através do scanner

Em relação à conexão segura, avaliou-se que o certificado do *e-commerce real* é homologado por *VeriSign, Inc*⁴². A conexão é criptografada com AES-256. Esta dentro da validade e o nome do site coincide com o nome apresentado no certificado. Em relação ao certificado, a configuração esta correta, porém o uso da conexão criptografada não é forçada, como apresentado anteriormente.

Concluindo a análise do *e-commerce real*, verificou-se que nenhum dos *cookies* usados pela loja virtual usam os atributos *HttpOnly* e *secure*. Como já abordado no trabalho, estes atributos reforçam a segurança dos *cookies*.

De forma geral pode-se observar que o *e-commerce real* mostrou-se bastante inseguro. Tal situação é bastante perigosa, pois tal loja virtual é bastante conhecida e possui muitos clientes. E a avaliação de segurança feita por terceiros não revelou as fraquezas apontadas por este trabalho, passando a sensação de falsa segurança aos usuários. A Tabela 2 apresenta um resumo da avaliação feita no *e-commerce real*.

Tabela 2: Avaliação do *e-commerce real*.

Característica	<i>e-commerce real</i>
Transporte seguro (HTTPS)	Sim, porém não é forçado
Arquivo robots.txt	Sim, porém sem restrições
Enumeração de usuários no login	Não
Enumeração de usuários no cadastro	Sim
Cookie criptografado	Sim
Cookies com atributos bem definidos e configurados	Não
Logout seguro (invalida o cookie)	Não
Timeout Logout	Não
Proteção CSRF	Não
Mecanismo para avaliar força da senha	Não
Mecanismo para detectar senhas comuns	Não
Geração segura de boletos	Não

⁴²<http://www.verisign.com.br/>

7.3 *E-commerce PrestaShop*

O *e-commerce* de código aberto escolhido foi o *PrestaShop* 1.4.8.2⁴³, desenvolvido usando a linguagem PHP e banco de dados *MySQL*.

Os testes de segurança foram feitos usando a instalação padrão do *PrestaShop* localmente, desta forma os testes foram realizados tanto no *front office*, área de exposição e compra de produtos, e também no *back office*, área de administração dos produtos, vendas e outros assuntos da loja. Testes que envolvem aplicações que já estão na Internet não foram considerados, pois como mencionado, o teste foi realizado na aplicação instalada localmente.

7.3.1 Teste do *e-commerce PrestaShop*

Detalhes importantes em relação a segurança são apresentados logo na instalação do *e-commerce PrestaShop*. Ao final do processo de instalação, duas dicas de segurança são sugeridas: (1) remoção da pasta `install`; e (2) alteração do nome do diretório do *back office*. A Figura 11 ilustra estas dicas.

Assim que a instalação da loja virtual termina é necessário remover a pasta `install`, já que é bem provável que esses arquivos não sejam mais usados. Em relação a pasta `admin`, que contém os arquivos que fazem parte da administração da loja, é preciso renomeá-la. Como a loja virtual *PrestaShop* é um *software* de código aberto, um atacante pode analisar todos os códigos e tomar todos os conhecimentos necessários para explorar a loja apenas instalando-a localmente. Assim em um primeiro momento ele irá conhecer a aplicação em uma instalação local e após isso parte para um ataque real em uma loja em produção na Internet. Ao

⁴³<http://www.prestashop.com/en/download>

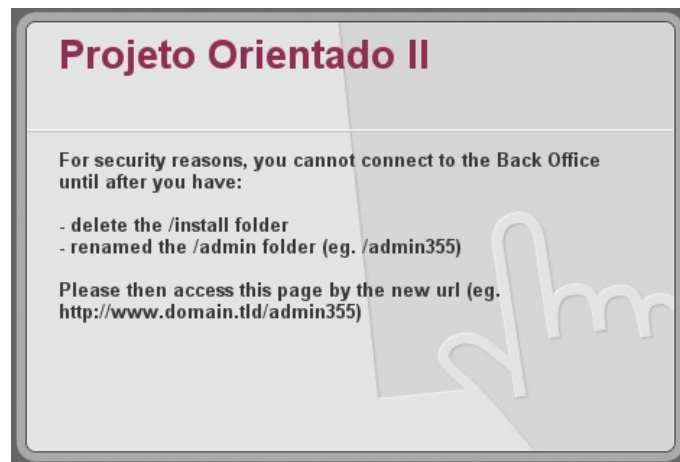


Figura 11: *E-commerce PrestaShop*: dicas de segurança ao final da instalação

renomear a pasta da administração, técnica de segurança por obscuridade, o usuário estará fugindo do nome padrão que comumente é usado em diversas aplicações *Web*. Vale notar que, se estas duas dicas de segurança não forem aplicadas, o acesso a área administrativa não é concedido.

Outro detalhe em relação a instalação padrão, é que não existe nenhum arquivo para controlar a indexação feita pelos mecanismos de buscas, isto é o arquivo `robots.txt`.

Os testes que seguem referem-se ao *back office* da loja virtual.

O mecanismo de *login* não oferece indícios de qual dado fornecido está incorreto, e-mail ou senha. A mensagem apresentada é ilustrada pela Figura 12.

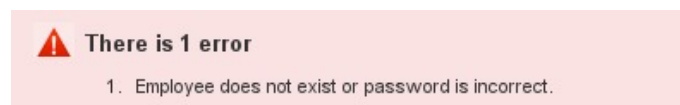


Figura 12: *E-commerce PrestaShop*: mensagem informando credenciais erradas

Ao enviar um e-mail correto e uma senha incorreta, a aplicação lança a mensagem que existe um erro, porém não especifica onde o erro está. De acordo com a metodologia de testes da *OWASP*, essa é uma prática a ser seguida.

Em nenhum momento da instalação da loja virtual foi concedido uma credencial padrão para acesso a área de administração da loja. O usuário deve preencher um formulário fornecendo um e-mail e uma senha para ser a credencial de acesso ao *back office*. Desta forma não existe contas padrão para acesso inicial e/ou configuração.

Ao acessar a página de *login*, um *cookie* com o nome `0ce3beb550fcd80fa4e-2751c91d13df6` foi criado. Após inspeção de código foi possível descobrir que tal nome é na verdade o valor `psAdmin` criptografado⁴⁴ usando MD5. Desta forma referências ao *cookie* `0ce3beb550fcd80fa4e2751c91d13df6` serão feitas usando o nome sem criptografia, ou seja, `psAdmin`. A inspeção de código foi feito apenas para descobrir o nome do *cookie* para melhor referenciá-lo no trabalho.

Antes da autenticação, o *cookie* `psAdmin` possuía 126 *bytes*. Após autenticação bem sucedida, o *cookie* tem seu valor alterado e conseqüentemente seu tamanho passa para 274 *bytes*. Novas informações foram adicionadas, porém são criptografadas. Devido a criptografia do *cookie* usado, não foi possível realizar a modificação de parâmetros para avaliar o mecanismo de autenticação.

Após acionar o mecanismo de *logout*, o *cookie* `psAdmin` usado pela aplicação teve seu valor restaurado para o valor de antes da autenticação, ou seja, retornou ao tamanho 126 *bytes*. Também foi possível verificar que a aplicação não conta

⁴⁴Na linha 66 do arquivo `classes/Cookie.php` o comando `md5` é usado na variável `name`. Em relação ao *back office*, na linha 33 do arquivo `/<caminho_admin>/init.php` um novo *cookie* é criado e o nome passado como parâmetro é `psAdmin`.

com *timeout logout*, ou seja, o administrador não é deslogado automaticamente após certo tempo de inatividade.

No momento em que o administrador estava autenticado no *back office*, o valor do *cookie* `psAdmin` foi salvo para avaliar a possibilidade de retornar ao painel administrativo apenas com seu valor. Assim acionou-se o mecanismo de *logout* e restaurou-se o valor do *cookie*. Feito isso, várias tentativas de retornar a área administrativa da loja virtual foram feitas. Nenhuma com sucesso, isto implica que existe algum mecanismo que invalida o valor do *cookie* após o *logout* do usuário. Ou seja, não é possível acessar o *back office* apenas com o *cookie*.

Sucessivas autenticações bem sucedidas foram realizadas afim de avaliar o *cookie* `psAdmin`. O valor era sempre modificado em *logins* e *logouts*, porém devido a criptografia de seu valor impossibilitou a avaliação do que realmente esta sendo alterado.

Os URLs do *back office* possuem *tokens*, o que dificulta o ataque CSRF. Se um *token* inválido for detectado pela aplicação, uma página de erro é apresentada ao usuário. A Figura 13 mostra a mensagem da página de erro.



Figura 13: *E-commerce PrestaShop*: mensagem informando um erro de *token*

Mesmo o *token* inválido sendo detectado, a aplicação ainda oferece a opção de acessar a página.

Para que o ataque CSRF possa ser bem sucedido, o atacante precisa saber a estrutura dos URLs e então usar algum URL que executa uma ação sensível na aplicação. Tal URL será enviado ao usuário que, executará tal ação sem consentimento. Da forma como é implementado no *e-commerce PrestaShop*, não basta apenas o atacante conhecer a estrutura dos URLs, ele ainda precisa ter o token válido, caso contrário a página de erro será apresentada ao usuário.

Os testes que seguem referem-se ao *front office* da loja virtual.

Ao acessar o *front office*, um *cookie* com o nome `8812c36aa5ae336c2a77bf-63211d899a` é adicionado. Da mesma forma feito para o *back office*, pode-se verificar através do código que o valor `8812c36aa5ae336c2a77bf63211d899a` é na verdade o valor `ps` criptografado com MD5. Para melhor referenciá-lo será usado o nome `ps`.

Em um primeiro acesso, o valor do *cookie* `ps` tem tamanho *190 bytes* e seu valor é criptografado. Após *login* bem sucedido, o tamanho do *cookie* passa para *422 bytes*. Após *logout* o *cookie* retorna ao tamanho *190 bytes*. Conclui-se que informações que garante a autenticação do usuário foram adicionadas ao *cookie*, porém como seu valor é criptografado não foi possível verificar quais informações foram adicionadas.

O valor do *cookie* `ps` foi salvo quando o usuário estava logado na loja virtual. Após isto, o mecanismo de *logout* foi acionado e então o valor do *cookie* foi restaurado para o valor de quando ele estava autenticado na aplicação. Ao tentar requisitar uma página que necessitava de autenticação, a aplicação executou a ação de redirecionamento para a página de *login*. Com isto pode-se observar que existe algum mecanismo ou valor no *cookie* que permite a aplicação distinguir um *cookie*

devidamente válido ou não. Como tais valores são criptografados, não foi possível verificar.

Na página de autenticação não foi possível enumerar e-mails válidos na loja virtual. Ao fornecer credenciais inválidas, a mensagem apresentada era de que havia um erro e a autenticação falhou. A Figura 14 mostra a mensagem de erro de autenticação no *front office* do *e-commerce PrestaShop*.

There is 1 error :
01. Authentication failed

Figura 14: *E-commerce PrestaShop*: mensagem informando credenciais erradas

Nenhum detalhe extra, que pode ser usado para identificar se o e-mail ou senha estão errados, foi fornecido.

Assim como no *back office* do *PrestaShop*, não foi possível ignorar o esquema de autenticação, seja realizando requisição direta de página ou modificando parâmetros.

Se o usuário fechar o navegador e depois abrir novamente, ele ainda permanecerá autenticado na loja virtual. Tentou-se copiar o *cookie ps* e usá-lo em outro navegador para verificar se era possível acessar as informações do cliente apenas com o *cookie ps*. Ao copiar o *cookie* e tentar acessar a conta do cliente na loja, a aplicação fez um redirecionamento para a página de *login*. Isto permite verificar, novamente, que a aplicação tem um controle rígido sobre os valores do *cookie*.

Assim como no *back office*, o *front office* não tem mecanismo de *logout* automaticamente.

A enumeração de e-mails válidos na aplicação pode ser feita na página de cadastro de usuários. Antes de avaliar se o e-mail fornecido já está associado a algum

cliente, a aplicação valida todos os outros dados fornecidos e por último avalia o e-mail informado. Caso já exista associação, uma mensagem de erro é lançada informando a situação. Não existe nenhum mecanismo para evitar tentativas automáticas de enumeração de e-mails válidos, como CAPTCHA ou mecanismo para bloquear tentativas erradas e seguidas.

Ainda na página de cadastro de clientes, foi possível avaliar que o campo senha exige no mínimo cinco caracteres. Não existe nenhuma regra para a criação da senha, como uso de letras e dígitos ou impossibilidade de usar seu nome ou e-mail como senha. Bem como não existe nenhum mecanismo para avaliar a força da senha e conseqüentemente barrar senhas fracas no momento do cadastro. O fato de ser possível enumerar e-mails válidos e a senha mínima exigida ser cinco caracteres pode motivar um atacante a tentar um ataque de força bruta. É bem possível obter sucesso com este ataque nestas condições.

Tanto no *back office* quanto no *front office*, os *cookies* possuem os atributos *domain* e *path* bem definidos para a loja em questão. A configuração usada não permite que um *cookie* de uma loja no mesmo domínio porém em caminho diferente usem o mesmo *cookie*. Além disso o atributo *HttpOnly* também é empregado. A Tabela 3 apresenta um resumo da avaliação feita no *e-commerce PrestaShop*.

7.4 E-commerce básico

O terceiro *e-commerce* testado foi desenvolvido por Souza (2012). O sistema foi desenvolvido em *Ruby-on-Rails* seguindo a metodologia de desenvolvimento seguro de aplicações *Web* da *OWASP* (CURPHEY *et al.*, 2005). As funcionalidades mais básicas para realização de compras seguras na Internet foram implementadas neste *e-commerce*.

Tabela 3: Avaliação do *e-commerce PrestaShop*.

Característica	<i>e-commerce PrestaShop</i>
Arquivo robots.txt	Não
Enumeração de usuários no login	Não
Enumeração de usuário no cadastro	Sim
Credências padrões	Não
Cookie criptografado	Sim
Cookies com atributos bem definidos e configurados	Sim
Logout seguro (invalida o cookie)	Sim
Timeout Logout	Não
Proteção CSRF	Sim
Mecanismo para avaliar força da senha	Não
Mecanismo para detectar senhas comuns	Não

Uma descrição com maior detalhamento sobre as decisões tomadas e metodologias seguidas na implementação do *e-commerce básico* pode ser encontrado em (SOUZA, 2012).

O teste desta aplicação tem como objetivo validar o guia de desenvolvimento seguro da *OWASP*, já que tal *e-commerce básico* foi desenvolvido seguindo estritamente tal metodologia.

Assim como o *e-commerce PrestaShop*, este também foi testado localmente, portanto nem todos os testes descritos na metodologia da *OWASP* foram aplicados.

7.4.1 Teste do *e-commerce básico*

Os testes que seguem referem-se ao *front office* do *e-commerce básico*.

O sistema de autenticação da aplicação usa CPF e senha como credenciais para acesso à loja virtual. Tal sistema não fornece informações suficientes que possibilitem a coleta de diversos usuários válidos da aplicação. A mensagem de

erro é única: “*Dados incorretos*”, não sendo possível distinguir se o CPF estava correto ou não.

A aplicação faz uso de um *cookie* durante toda navegação. Ao acessar pela primeira vez, o *cookie _P02_session* foi criado, inicialmente com o tamanho de 230 *bytes*. Após autenticação bem sucedida o *cookie* passou a ter o tamanho de 470 *bytes*. Não foi possível verificar com certeza quais são os valores armazenados no *cookie* pois seu valor é criptografado.

A partir de uma navegação pela loja e observação de seu tamanho, foi possível verificar que além das informações que fazem controle do usuário autenticado, as informações da compra tais como quantidade e produtos, também eram armazenadas neste *cookie*. Uma observação cabível em relação a este modo de uso dos *cookies* é que existe um valor máximo de tamanho para os *cookies*. No *e-commerce básico*, se a compra for muito grande, ou seja, vários itens diferentes, corre o risco da aplicação não funcionar como esperado, já que os valores do carrinho são armazenados em um *cookie* criptografado. A criptografia aumenta o tamanho dos dados e com vários itens o tamanho fica relativamente grande para um *cookie*.

A partir da situação de cliente autenticado e com o carrinho de compras com produtos, o *cookie _P02_session* foi alterado, mesmo não sabendo o sentido da alteração. O comportamento da aplicação foi invalidar a sessão do usuário, deslogando-o e redirecionando-o para a página inicial da loja virtual.

A partir da autenticação bem sucedida, o valor do *cookie* foi salvo e então em outro navegador *Web* tentou-se usar tal *cookie* para obter acesso à conta do usuário previamente logado em outro navegador. A aplicação invalidou o *cookie* e a tentativa foi fracassada pois o usuário foi redirecionado para a página de autenticação.

Da mesma forma, salvou-se o valor do *cookie* e acionou-se o mecanismo de *logout*. Tentativas de restaurar o *cookie* foram feitas para obter acesso à conta do usuário. Porém pode-se certificar que a aplicação invalidou o *cookie* e não foi possível voltar a conta do cliente com tal *cookie*. Isto implica que o controle de usuário logados não é feito apenas com *cookies* e que a aplicação de fato invalida o *cookie* e não apenas o descarta.

Ainda em relação ao mecanismo de autenticação, foi possível verificar que a aplicação não aplica nenhum bloqueio após certo número de tentativas erradas. Isto pode conduzir um atacante a realizar um ataque de força bruta.

O *cookie _P02_session* que é bastante usado na aplicação possui seu valor criptografado. Ele também faz uso do atributo *HttpOnly* que impossibilita o manuseio através de linguagens de scripts.

Os testes que seguem referem-se ao *back office* do *e-commerce básico*.

A credencial para acesso a área administrativa da loja virtual foi disponibilizada afim de estender o teste também ao *back office*.

A página de gerenciamento de clientes possui as funcionalidades de ‘mostrar’, ‘editar’ e ‘deletar’ clientes. Para cada uma dessas ações, uma variável *id* é especificada afim de distinguir cada cliente. Pode-se verificar que não houve verificação do tipo da entrada fornecida. A *id* na aplicação é apenas um número inteiro e tal verificação não foi feita. Assim tentativas de injeção de códigos SQL para exibição de todos clientes e para remoção de todos clientes foram feitas. Apesar da falta de validação, a aplicação não apresentou vulnerabilidades nesta variável *id*, não sendo possível listar nem deletar nenhum cliente da loja.

A ação de remover usuários é confirmada com um *popup*. Esta alternativa é uma das táticas para evitar o ataque CSRF, pois o usuário não irá executar a ação de remover usuários sem tomar conhecimento do que está sendo feito. A ação não é executada diretamente, caso um atacante faça um administrador da loja executar a ação de deletar usuários, uma janela será lançada para a confirmação da ação. Apesar de não ser o método mais eficiente para a proteção contra ataques CSRF, a aplicação apresenta uma alternativa.

De forma geral, o *e-commerce básico* apresentou bom comportamento em relação a proteção dos dados do usuário e suas compras. A aplicação possui um controle bem rígido dos usuários autenticados. O objetivo final foi alcançado, pois foi possível certificar que o guia de desenvolvimento seguro de aplicações *Web* da *OWASP* de fato norteia os desenvolvedores nos aspectos de proteção de dados sensíveis e boas práticas de segurança. A Tabela 4 apresenta um resumo da avaliação feita no *e-commerce básico*.

Tabela 4: Avaliação do *e-commerce básico*.

Característica	<i>e-commerce básico</i>
Arquivo robots.txt	Não
Enumeração de usuários no login	Não
Enumeração de usuário no cadastro	Sim
Credências padrões	Não
Cookie criptografado	Sim
Cookies com atributos bem definidos e configurados	Sim
Logout seguro (invalida o cookie)	Sim
Timeout Logout	Não
Proteção CSRF	Sim
Mecanismo para avaliar força da senha	Não
Mecanismo para detectar senhas comuns	Não

7.5 Comentários finais

A partir das análises apresentadas é possível notar que as três aplicações possuem fraquezas similares. A enumeração de usuários é possível em todas elas e pode ser feita de maneira automática, pois nenhuma delas possui algum mecanismo para diferenciar ações humanas de ações automatizadas, um CAPTCHA por exemplo. Uma possível desculpa para a não utilização de tal mecanismo é que dificultaria o processo de cadastro de usuários. Porém é necessário para garantir a segurança da aplicação.

Outro fato que todas apresentam em comum é que não existe um mecanismo para avaliar a força das senhas dos usuários. E também para detectar senhas fracas ou bastante usadas. Um alerta no momento do cadastro já faria diferença. Descuidos em relação a isso possibilitam e encorajam um atacante a tentar um ataque de força bruta.

A Tabela 5 mostra as características dos três *e-commerces* testados.

Tabela 5: Avaliação do *e-commerce real*, *e-commerce PrestaShop* e *e-commerce básico*.

Característica	Real	Presta	Básico
Transporte seguro (HTTPS)	Sim	–	–
Arquivo robots.txt	Sim	Não	Não
Enumeração de usuários no login	Não	Não	Não
Enumeração de usuários no cadastro	Sim	Sim	Sim
Credências padrões	–	Não	Não
Cookie criptografado	Sim	Sim	Sim
Cookies com atributos bem definidos e configurados	Não	Sim	Sim
Logout seguro (invalida o cookie)	Não	Sim	Sim
Timeout Logout	Não	Não	Não
Proteção CSRF	Não	Sim	Sim
Mecanismo para avaliar força da senha	Não	Não	Não
Mecanismo para detectar senhas comuns	Não	Não	Não
Geração segura de boletos	Não	–	–

8 CONCLUSÃO

Este trabalho abordou as aplicações *Web* como uma nova plataforma de execução de tarefas. Mostrou-se que uma mesma aplicação, instalada em algum servidor, pode ser usada por vários usuários aos mesmo tempo. Elas podem realizar diversas tarefas e lidar com dados pessoais de muitos usuários. Devido ao fato de uma mesma aplicação poder ser usada por muitos usuários e manusear seus dados, foi proposto uma avaliação para examinar a segurança de tais aplicações.

As aplicações escolhidas foram os *e-commerces*, pois para seu uso, os usuários precisam informar diversas informações como: nome, CPF, RG, endereço completo e dados de cartões de crédito. A natureza dessas informações é bastante crítica, pois se a aplicação falhar esses dados podem vazar na Internet e tornarem-se públicos.

Desta forma, o objetivo do trabalho foi avaliar aplicações *Web* em relação à segurança. Este objetivo foi alcançado após estudo e apresentação da metodologia de testes de segurança de aplicações *Web* da *OWASP*. Com uso desta metodologia, três aplicações de *e-commerce* foram colocadas à prova.

A partir das análises e discussões feitas sobre os *e-commerces* testados, pode-se verificar que existem aplicações reais que passam a sensação de falsa segurança aos seus clientes. Elas fazem isso disponibilizando selos e avisos que tal aplicação foi severamente testada e que os dados não estão vulneráveis. Avisos estes que não se mostraram reais após os testes executados.

O trabalho mostrou-se importante na área de desenvolvimento de aplicações e segurança computacional pois a partir da avaliação de um *e-commerce* real, foi possível verificar diversas falhas e descuidos em relação aos dados de clientes. De

forma geral a principal contribuição foi chamar atenção de usuários e desenvolvedores de aplicações *Web* em relação as consequências das falhas de aplicações *Web*, principalmente das lojas virtuais.

Como trabalho futuro pode-se tomar o guia de revisão de códigos – *Code Review Guide* escrito por Stock *et al.* (2008) e então aplicá-lo em aplicações reais. Tal guia tem como objetivo realizar uma auditoria segura de códigos para determinar vulnerabilidades.

Referências

ALONSO, G.; CASATI, F.; KUNO, H.; MACHIRAJU, V. *Web Services: Concepts, Architectures and Applications*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 3642078885, 9783642078880.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I. *et al.* A view of cloud computing. *Communications of the ACM*, ACM, v. 53, n. 4, p. 50–58, 2010.

AUGER, R.; BARNETT, R. The wasc threat classification v2.0. *The Web Application Security Consortium*, 2010. Disponível em: <[http://projects-webappsec.org/w/page/13246978/Threat%20Classification](http://projects.webappsec.org/w/page/13246978/Threat%20Classification)>.

BARTO, J. *OWASP Application Security Metrics Project*. 2009. Disponível em: <https://www.owasp.org/index.php/Category:OWASP_Legal_Project>.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. *Hypertext Transfer Protocol – HTTP/1.0*. IETF, maio 1996. RFC 1945 (Informational). (Request for Comments, 1945). Disponível em: <<http://www.ietf.org/rfc/rfc1945.txt>>.

BERNERS-LEE, T.; MASINTER, L.; MCCAHERN, M. *Uniform Resource Locators (URL)*. IETF, dez. 1994. RFC 1738 (Proposed Standard). (Request for Comments, 1738). Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986, 6196, 6270. Disponível em: <<http://www.ietf.org/rfc/rfc1738.txt>>.

BHATNAGAR, S. *Textbook of Computer Science: for Class XII*. Prentice-Hall of India Pvt.Ltd, 2008. ISBN 812033518X. Disponível em: <<http://books.google.com.br/books?id=jRobT0OWfVsC>>.

- BODMER, F. Cross-site scripting. 2007.
- CHADWICK, J. *Programming Razor*. O'Reilly Media, 2011. ISBN 1449306764.
Disponível em: <<http://books.google.com.br/books?id=2l-GvnrR9AIC>>.
- CURPHEY, M.; ENDLER, D.; HAU, W.; TAYLOR, S.; SMITH, T.; RUSSELL, A.; MCKENNA, G.; PARKE, R.; MCLAUGHLIN, K.; TRANTER, N. *et al.*
A guide to building secure web applications and web services. *The Open Web Application Security Project*, v. 1, 2005.
- DOCUMENT Object Model (DOM). 2005. Disponível em: <<http://www.w3.org/DOM/>>.
- FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T. *Hypertext Transfer Protocol – HTTP/1.1*. IETF, jun. 1999. RFC 2616 (Draft Standard). (Request for Comments, 2616). Updated by RFCs 2817, 5785, 6266. Disponível em: <<http://www.ietf.org/rfc/rfc2616.txt>>.
- FIELDS, D. K.; KOLB, M. A.; BAYERN, S. *Web Development with Java Server Pages*. 2nd. ed. Greenwich, CT, USA: Manning Publications Co., 2001. ISBN 193011012X.
- GARRETT, J. *et al.* Ajax: A new approach to web applications. *Adaptive path*, v. 18, 2005.
- GOODMAN, D.; MORRISON, M.; NOVITSKI, P.; RAYL, T. G. *JavaScript Bible*. 7th. ed. [S.l.]: Wiley Publishing, 2010. ISBN 0470526912, 9780470526910.
- GORDEYCHIK, S.; GROSSMAN, J.; KHERA, M.; LANTINGA, M.; WYSOPAL, C.; ENG, C.; SHAH, S.; LEE, L.; MURRAY, C.; EVTEEV, D.
Web application security statistics. *The Web Application Security Consortium*,

2008. Disponível em: <<http://projects.webappsec.org/w/page/13246989-/Web%20Application%20Security%20Statistics>>.

GROSSMAN, J. Cross site tracing (xst). *WhiteHat Security White Paper*, 2003.

GUNDAVARAM, S. *CGI programming on the World Wide Web*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1996. ISBN 1-56592-168-2.

GUTTMAN, B.; ROBACK, E. *An introduction to computer security: the NIST handbook*. [S.l.]: DIANE Publishing, 1995.

HAAS, H. Web services glossary. 2004. Disponível em: <<http://www.w3.org/TR/ws-gloss/>>.

HOFFMAN, B. Ajax security dangers. *SPI Dynamics*, 2006.

HOWARD, M.; LEBLANC, D. *Writing Secure Code*. Redmond, WA, USA: Microsoft Press, 2001. ISBN 0735615888.

JACOBS, I.; RAGGETT, D.; HORS, A. L. *HTML 4.01 Specification*. [S.l.], dez. 1999. [Http://www.w3.org/TR/1999/REC-html401-19991224](http://www.w3.org/TR/1999/REC-html401-19991224).

KESTEREN, A. van. Xmlhttprequest. *Retrieved June*, 2010.

KIM, W. Cloud computing: Today and tomorrow. *Journal of Object Technology*, v. 8, n. 1, p. 65–72, 2009.

KLEIN, A. Divide and conquer. *HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics, Sanctum whitepaper*, 2004.

KUROSE, J.; ROSS, K. *Redes de Computadores E Internet*. [S.l.]: Pearson Education, 2006. ISBN 8478290834.

LEHTINEN, R.; RUSSELL, D.; GANGEMI, G. T. *Computer Security Basics*.

[S.l.]: O'Reilly Media, Inc., 2006. ISBN 0596006691.

MALL, R. *Fundamentals of Software Engineering*. PHI Learning,

2009. ISBN 8120338197. Disponível em: <[http://books.google.com.br-](http://books.google.com.br-books?id=CFhtgKrKL1EC)

[/books?id=CFhtgKrKL1EC](http://books?id=CFhtgKrKL1EC)>.

MEUCCI, M. Owasp testing guide version 3.0. *OWASP Foundation, November,*

2008.

MITNICK, K. D.; SIMON, W. L. *The Art of Deception: Controlling the Human*

Element of Security. New York, NY, USA: John Wiley & Sons, Inc., 2001. ISBN

1401463223.

NEWMAN, R. *Computer Security: Protecting Digital Resources*. USA: Jones

and Bartlett Publishers, Inc., 2009. ISBN 0763759945, 9780763759940.

NIEMEYER, P.; KNUDSEN, J. *Learning Java*. 3rd. ed. Sebastopol, CA, USA:

O'Reilly & Associates, Inc., 2005. ISBN 9780596008734.

PAKALA, S.; NINER, P. *OWASP AppSec FAQ*. 2011. Disponível em:

<https://www.owasp.org/index.php/OWASP_AppSec_FAQ>.

PETER, I. The beginnings of the internet. *Internet: [http://www. nethistory.](http://www.nethistory.info/History%20of%20the%20Internet/beginnings.html)*

info/History% 20of% 20the% 20Internet/beginnings. html, Accessed, v. 30, n. 10,

p. 2007, 2004.

PFLEEGER, C. P.; PFLEEGER, S. L. *Security in Computing (4th Edition)*. Upper

Saddle River, NJ, USA: Prentice Hall PTR, 2006. ISBN 0132390779.

ROBINSON, D.; COAR, K. *The Common Gateway Interface (CGI) Version 1.1*. IETF, out. 2004. RFC 3875 (Informational). (Request for Comments, 3875). Disponível em: <<http://www.ietf.org/rfc/rfc3875.txt>>.

SHAH, S. Hacking web 2.0 applications with firefox. 2006. Disponível em: <<http://www.symantec.com/connect/articles/hacking-web-20-applications-firefox>>.

SHAW, W. *Cybersecurity for SCADA Systems*. PennWell Corporation, 2006. ISBN 1593700687. Disponível em: <<http://books.google.com.br/books?id=EyZVJ8KI8C0C>>.

SHIFLETT, C. Security corner: Session fixation. *Reprint of article from phpl architect*, 2004.

SOARES, L.; LEMOS, G.; COLCHER, S. Redes de computadores: das lans mans e wans às redes atm. *Editora Campus, Rio de Janeiro*, 1995.

SOUZA, L. L. Desenvolvimento seguro de aplicações web seguindo a metodologia owasp. 2012.

SPELT, K. Blind sql injection. *SPI Dynamics Inc*, 2003.

STALLINGS, W. *Criptografia e segurança de redes: Princípios e práticas*. 4a. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008. ISBN 9788576051190.

STOCK, A. Van der; CRUZ, D.; CHAPMAN, J.; LOWERY, D.; KEARY, E.; MORANA, M.; ROOK, D.; PREGO, J. *OWASP Code Review Guide Version 1.1*. 2008.

STUTTARD, D.; PINTO, M. *The web application hacker's handbook: discovering and exploiting security flaws*. New York, NY, USA: John Wiley & Sons, Inc., 2007. ISBN 9780470170779.

UCHÔA, J.; CAMINHAS, W.; SANTOS, T. da M. Algoritmos imunoinspirados aplicados em segurança computacional: utilização de algoritmos inspirados no sistema imune para detecção de intrusos em redes de computadores. UFMG, 2009.

UCHÔA, J. Q. *Segurança Computacional*. 2. ed. [S.l.]: Gráfica Universitária/UFLA, 2005. 59 p.

WALTHER, S. *ASP.NET Unleashed*. 2. ed. [S.l.]: Pearson Education, 2003. ISBN 067232542X.

WATKINS, P. Cross-site request forgery. *Bugtraq, June*, v. 13, 2001. Disponível em: <<http://www.tux.org/~peterw/csrf.txt>>.

WILLIAMS, J. *OWASP Legal Project*. 2011. Disponível em: <https://www.owasp.org/index.php/Category:OWASP_Legal_Project>.

WILLIAMS, J.; WICHERS, D. Owasp top 10 – 2010: The ten most critical web application security vulnerabilities. *OWASP Foundation, April*, 2010.

XHTML.ORG. 2000. Disponível em: <<http://www.xhtml.org/>>.