

UNIVERSIDADE DO ESTADO DO RIO GRANDE NO NORTE – UERN
FACULDADE DE CIÊNCIAS EXATAS E NATURAIS – FANAT
DEPARTAMENTO DE INFORMÁTICA – DI

ALFREDO HENRIQUE ALVES DE CASTRO

**SWAP – UMA FERRAMENTA PARA *PENTEST* DE APLICAÇÕES WEB DE
FORMA SIMPLIFICADA**

MOSSORÓ - RN

2019

ALFREDO HENRIQUE ALVES DE CASTRO

**SWAP – UMA FERRAMENTA PARA *PENTEST* DE APLICAÇÕES WEB DE
FORMA SIMPLIFICADA**

Monografia apresentada à Universidade do Estado do Rio Grande do Norte como um dos pré-requisitos para obtenção do grau de bacharel em Ciência da Computação, sob orientação do Prof. Dr. Sebastião Emídio Alves Filho.

MOSSORÓ - RN

2019

**Catálogo da Publicação na Fonte.
Universidade do Estado do Rio Grande do Norte.**

C355s Castro, Alfredo Henrique Alves de
SWAP - UMA FERRAMENTA PARA PENTEST DE
APLICAÇÕES WEB DE FORMA SIMPLIFICADA. / Alfredo
Henrique Alves de Castro. - Mossoró, 2019.
66p.

Orientador(a): Prof. Dr. Sebastião Emídio Alves Filho.
Monografia (Graduação em Ciência da Computação).
Universidade do Estado do Rio Grande do Norte.

1. Segurança. 2. Informação. 3. Vulnerabilidade. 4.
Ameaças. 5. Testes de Invasão. I. Emídio Alves Filho,
Sebastião. II. Universidade do Estado do Rio Grande do
Norte. III. Título.

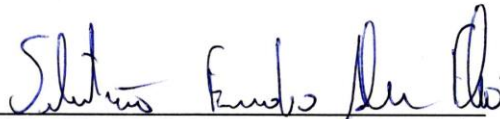
ALFREDO HENRIQUE ALVES DE CASTRO

SWAP: UMA FERRAMENTA PARA PENTEST DE APLICAÇÕES WEB DE FORMA SIMPLIFICADA

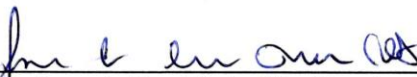
Monografia apresentada como pré-requisito para a obtenção do título de Bacharel em Ciência da Computação da Universidade do Estado do Rio Grande do Norte – UERN, submetida à aprovação da banca examinadora composta pelos seguintes membros:

Aprovada em: 10/05/2019

Banca Examinadora



Prof. Dr. Sebastião Emídio Alves Filho (Orientador)
Universidade do Estado do Rio Grande do Norte - UERN



Prof. Dr. Isaac de Lima Oliveira Filho (Examinador)
Universidade do Estado do Rio Grande do Norte - UERN



Prof. Alysson Mendes de Oliveira (Examinador)
Universidade do Estado do Rio Grande do Norte - UERN

AGRADECIMENTOS

Primeiramente à Deus por ter me concedido vida, saúde e forças para continuar a cada passo dado.

Aos meus pais, Lúcia e Luiz, que me deram sustento, condições e me apoiaram a cada decisão tomada.

À minha namorada e melhor amiga Ariele Melo que me ouviu quando reclamei, me consolou quando entristeci, me acalmou quando estressei e não me deixou desistir quando me senti pressionado.

Aos meus melhores amigos Thalison e Júnior, que me apoiaram por todos os anos que nos conhecemos.

Aos professores da universidade que durante o curso passaram seus ensinamentos, em especial à dois deles, Sebastião e Alysson, que além do conhecimento passado, transmitiram seus valores e paixões pela área. E ao professor Isaac por aceitar fazer parte da banca examinadora.

Aos professores durante o curso técnico integrado em informática que me fizeram crescer, amadurecer e ter outra visão para a área, em especial à Felipe, Cleone, Galba e Rodrigo, que foram, além de professores, verdadeiros amigos.

Aos amigos do meu antigo grupo Paulo Isaias, Allan Kardecc, Daniel, Judson, Israel, Alex, Ricardo, Kergionaldo, Samuel, Sadrak, Paulo, Jório (*in memoriam*) e Hugo (*in memoriam*) por todos os momentos bons e ruins na vida.

Aos meus amigos de faculdade Mateus, Veridiano, Francisco Vitor, Davi e André que me acolheram e formaram o melhor grupo possível de se ter durante um curso. E à Exlley, Vitor e Hitalo com quem morei junto por seis meses.

Aos meus amigos de ônibus Márcio, Pedro, Eilson, Sabino, Henrique, Geovanny, Tiago, Paulo Henrique e Fernando por me ajudarem na adaptação no início e por todas as risadas e momentos engraçados durante as viagens.

Aos meus amigos do grupo de segurança Kauê, Gustavo e Fernando por me incentivarem a conhecer a área de segurança da informação.

"Tudo o que a sua mão encontrar para
fazer, faça-o com todo o seu coração
[...]"

- Salomão

RESUMO

Com a popularização da informação, o avanço das tecnologias e do acesso à internet, houve, de maneira descontrolada, uma grande exposição dessas informações. A segurança da informação objetiva desenvolver soluções que combatam as consequências desse tipo de exposição, logo que cada vez mais as organizações solicitam que os sistemas de informação sejam seguros e robustos a fim de não serem prejudicados. Os testes de invasão foram desenvolvidos no intuito de antecipar a identificação das falhas nesses sistemas. Este trabalho propõe uma ferramenta que execute esses testes de maneira simplificada e automatizada, visando ser utilizada tanto pelos profissionais da área como pelos usuários comuns. Essa ferramenta conta com quatro módulos responsáveis por mapear os sistemas web, extrair de suas páginas os formulários e campos de entrada, injetar nesses campos cargas maliciosas que explorem as falhas e por fim gerar um *log* contendo as vulnerabilidades encontradas. Sendo testada em um ambiente controlado, uma máquina virtual construída com sistemas vulneráveis, apropriados para testes manuais e automatizados.

Palavras-chave: Segurança, Informação, Vulnerabilidade, Ameaças, Testes de Invasão.

ABSTRACT

With the popularization of information, the advancement of technologies and the access to the internet there was, in an uncontrolled way, a big exposition of this information. The information security seeks to develop solutions that fight against the consequences of these expositions, the organizations are looking for the most secure and tough information systems in order not to be harmed. The penetration tests were created with the goal to anticipate the identification of these systems' flaws, grounding this work, which proposes a tool that executes these tests in an easy and automatized way, seeking to be used by security professionals and ordinary users. This tool counts with four modules that are responsible for map the whole web system, extract from it pages their forms and input fields, inject in these fields payloads that explores it flaws and generate a log, which contains the vulnerabilities found. Being tested in a controlled environment, a virtual machine built with vulnerable systems, suitable for manual and automatized tests.

Keywords: Security, Information, Vulnerability, Threats, Pentest.

LISTA DE SIGLAS

ABNT	–	Associação Brasileira de Normas Técnicas
API	–	<i>Application Programming Interface</i>
CIA	–	Confidencialidade, Integridade, Disponibilidade
DOM	–	<i>Document Object Model</i>
DOS	–	<i>Denial of Service</i>
DVWA	–	<i>Damn Vulnerable Web Application</i>
HTML	–	<i>Hypertext Markup Language</i>
HTTP	–	<i>Hypertext Transfer Protocol</i>
ITU	–	<i>International Telecommunication Union</i>
OWASP	–	<i>Open Web Application Security Project</i>
RFC	–	<i>Request for Comments</i>
SI	–	Segurança da Informação
SPA	–	<i>Single Page Application</i>
SQL	–	<i>Structured Query Language</i>
SQLIA	–	<i>SQL Injection Attacks</i>
SWAP	–	<i>Simple Web Application Pentest</i>
XSS	–	<i>Cross-Site Scripting</i>
ZAP	–	<i>Zed Attack Proxy</i>

LISTA DE FIGURAS

Figura 1 - Fluxo de ação da exploração de uma vulnerabilidade.....	20
Figura 2 - Estrutura de um ataque passivo.	21
Figura 3 - Estrutura de um ataque ativo	23
Figura 4 - Tela inicial da ferramenta ZAP do grupo OWASP.....	26
Figura 5 - Ciclo de execução de um <i>pentest</i>	27
Figura 6 - Página web com campo vulnerável à XSS Refletido.....	36
Figura 7 - Campo vulnerável à XSS Refletido preenchido com valores comuns e refletindo o valor sem nenhum tipo de tratamento.....	36
Figura 8 - Página web após injeção de script.....	37
Figura 9 - Página web que permite o envio de comentários.....	38
Figura 10 - Ataque XSS Persistente executado com sucesso.....	38
Figura 11 - Página web que utiliza o valor do ID para consultas ao banco de dados.	41
Figura 12 - Injeção de SQL do tipo tautologia no campo ID.	41
Figura 13 – Diagrama de componentes apresentando a estrutura dos módulos da ferramenta e suas dependências.....	46
Figura 14 - Fluxograma de funcionamento da ferramenta SWAP.	47
Figura 15 - Funcionamento básico das requisições HTTP.	49
Figura 16 - Exemplo de funcionamento de um web scrapping.....	51
Figura 17 - Funcionamento da classe Scorpion.	54
Figura 18 - Interface inicial da ferramenta SWAP.	55
Figura 19 - Tela inicial do sistema <i>Mutillidae</i> da OWASP.....	58
Figura 20 - Tela Inicial do sistema DVWA.....	58
Figura 21 - Resultado dos testes XSS realizados pela ferramenta SWAP no sistema DVWA.	60
Figura 22 – Teste XSS manual executados no sistema DVWA.	61

LISTA DE ALGORITMOS

Algoritmo 1 - Pseudocódigo de funcionamento da ferramenta SWAP.....	46
Algoritmo 2 - Funcionalidade principal da classe <i>Knocker</i>	50
Algoritmo 3 - Funcionalidade que obtém os elementos HTML.	51
Algoritmo 4 - Funcionalidades para extração de Links e Formulários do sistema testado....	53

LISTA DE GRÁFICOS

Gráfico 1 - Número de vulnerabilidades em sistemas web entre 2016 e 2018.	28
Gráfico 2 - Total de incidentes reportados ao CERT.br por ano.	29
Gráfico 3 - Vulnerabilidades por sistema dos setores da indústria.	30
Gráfico 4 - Número de vulnerabilidades encontradas, entre de 2014 e 2018, seguindo a classificação da OWASP.	32

SUMÁRIO

1	INTRODUÇÃO	14
1.1.	Objetivos	15
1.2.	Metodologia.....	15
1.3.	Estrutura do trabalho	16
2	SEGURANÇA DA INFORMAÇÃO	17
2.1.	Definição	17
2.2.	Princípios.....	17
2.2.1.	Confidencialidade	18
2.2.2.	Integridade	18
2.2.3.	Disponibilidade	18
2.2.4.	Autenticidade	19
2.2.5.	Não-Repúdio ou Irretratabilidade.....	19
2.3.	Ameaças à segurança.....	19
2.3.1.	Ataques Passivos.....	21
2.3.2.	Ataques Ativos	22
2.4.	Segurança de sistemas web.....	24
2.4.1.	OWASP	25
2.5.	<i>Pentest</i>	26
3	OWASP Top Ten	28
3.1.	Contextualização.....	28
3.2.	Definição	30
3.3.	Evolução	33
3.4.	Vulnerabilidades.....	34
3.4.1.	<i>Cross-Site Scripting (XSS)</i>	34
3.4.1.1.	XSS Refletido	35
3.4.1.2.	XSS Persistente	37
3.4.1.3.	XSS baseados na DOM.....	39
3.4.1.4.	Contramedidas.....	39
3.4.2.	Injeção de código.....	39
3.4.2.1.	Classificação	42
3.4.2.2.	Contramedidas.....	43

4	SWAP – FERRAMENTA PARA <i>PENTEST</i> DE SISTEMAS WEB	44
4.1.	A Ferramenta	44
4.2.	Estrutura Geral	45
4.2.1.	Classe <i>Knocker</i>	48
4.2.2.	Classe <i>Claw</i>	50
4.2.3.	Classe <i>Sting</i>	52
4.2.4.	Classe <i>Scorpion</i>	52
4.2.5.	Interface Web	55
4.3.	Trabalhos Relacionados	55
5	TESTES E RESULTADOS	57
5.1.	Ambiente de testes	57
5.2.	Metodologia dos testes	59
5.3.	Resultados	59
6	CONSIDERAÇÕES FINAIS	62
	REFERÊNCIAS	64
	ANEXOS	66
	ANEXO A – Tabela de mudanças entre as duas últimas versões do OWASP Top Ten. ...	66

1 INTRODUÇÃO

Desde a última grande guerra, da qual a informação foi o componente mais significativo para o seu encerramento, esse elemento ganhou uma nova conotação e passou a possuir novos valores, que até então, não haviam sido agregados. As informações passaram a ser valiosas demais para os indivíduos, as organizações e, principalmente, para os governos. Além da própria guerra, os contextos culturais passaram a visualizar a informação de uma outra perspectiva, que a informação passou a ser universal e de alcance a todos, quebrando as barreiras geográficas que, antes do advento da rede mundial de computadores e dispositivos – a Internet, limitava o alcance dessas informações.

A Internet estabeleceu uma difusão de informações de maneira rápida, barata e acessível aos indivíduos e organizações. Porém essa dispersão pode ser utilizada tanto para fins benéficos, como publicações de materiais acadêmicos, quanto para fins maléficis, como a destruição ou roubo de informações com ou sem fins lucrativos. Essa comunicação rápida e barata possibilitou que indivíduos mal-intencionados pudessem agir de forma maliciosa e atentar contra as informações de ordem privada, causando o surgimento de fraudes virtuais que atinge tanto usuários comuns, quanto organizações comerciais, financeiras e governamentais.

Os sistemas de informação passaram a ser altamente visados por esses agentes maliciosos, que procuram brechas no funcionamento desses sistemas para que possam tirar proveito. A partir disso, estudos surgiram para estabelecer mecanismos de defesa contra a investida desses agentes, técnicas foram criadas para combater, identificar e expor os autores dessas investidas. Várias dessas técnicas foram desenvolvidas e muitas já se tornaram obsoletas, consequência do avanço e da maior complexidade dos ataques proferidos.

Surgiram assim também, técnicas que ajudam a identificar essas falhas de segurança e que possibilitam aos desenvolvedores dos sistemas de informação, trabalhar em cima dessas falhas de modo que haja uma contramedida na exploração delas. Essas técnicas ou conjunto de testes de invasão, se tornaram passo fundamental durante o desenvolvimento e/ou manutenção dos sistemas, pois sua utilização garante às organizações uma maior confiabilidade em seus *softwares*, repassando uma segurança maior aos usuários desses sistemas.

1.1. Objetivos

Este trabalho tem como objetivo apresentar um estudo acerca dessas ameaças e contramedidas abordadas, assim como o desenvolvimento de uma ferramenta que execute os testes de invasão apropriados, identifique nos sistemas web as falhas primárias que possibilitam a exploração indevida desses sistemas e das informações armazenadas, bem como construir um relatório explicitando essas falhas encontradas de maneira que possa ser lido por usuários técnicos e leigos.

1.2. Metodologia

Inicialmente foi realizado um levantamento bibliográfico da qual servisse de base para este trabalho, de maneira a auxiliar na construção de um referencial teórico sólido e coerente. De forma que permitisse ao trabalho utilizar-se dos conceitos levantados para tomar conhecimento das vulnerabilidades mais comuns, que traziam mais riscos aos sistemas. Visando construir uma ferramenta igualmente sólida e coerente.

Após o levantamento bibliográfico, foram pesquisadas ferramentas que executassem funções parecidas com a ferramenta proposta. De início encontrou-se ferramentas que exploram vulnerabilidades individualmente, mudando de metodologia de ferramenta para ferramenta e por fim foram encontradas ferramentas que centralizavam a maioria dos testes possíveis.

A implementação foi realizada com base em quatro módulos principais, seguindo os preceitos de uma das bibliografias levantadas. Esses quatro módulos são o núcleo de todo o sistema por serem responsáveis pela realização das tarefas mais importantes desempenhadas pela ferramenta.

Posterior a implementação, foram realizados testes que comprovam o funcionamento da ferramenta e medem o seu desempenho. Foi utilizado um ambiente controlado que oferecesse sistemas vulneráveis, propícios para serem testados e comprovarem o funcionamento da ferramenta.

Por último, foram observados os resultados gerados pela ferramenta em formato de página HTML, que reuniu todas as informações básicas do sistema, o

mapeamento de todo o sistema, os campos extraídos e testados pela ferramenta, os resultados das injeções de código maliciosos e os efeitos causados no sistema.

1.3. Estrutura do trabalho

O trabalho possui a seguinte organização:

No Capítulo 2 é apresentado uma contextualização acerca da área de segurança da informação, seus principais conceitos e áreas de atuação.

No Capítulo 3 é realizado um aprofundamento no estudo das ameaças e falhas que podem atingir os sistemas de informação, baseado no levantamento realizado por um grupo que atua na área de segurança da informação.

No Capítulo 4 é detalhado o desenvolvimento e funcionamento da ferramenta proposta.

No Capítulo 5 são abordados os testes executados pela ferramenta, sendo apresentado os resultados gerados, bem como a metodologia e o ambiente de testes.

O Capítulo 6 conclui o trabalho mostrando a relação da ferramenta desenvolvida com os conceitos abordados e com os objetivos traçados, as limitações durante o desenvolvimento e propõe trabalhos a serem desenvolvidos no futuro.

2 SEGURANÇA DA INFORMAÇÃO

Devido a crescente geração de dados e informações sigilosas por parte das organizações, a Segurança da Informação (SI) é uma área da computação em crescimento. Nesse capítulo são abordados os principais conceitos em torno dessa área, do qual este trabalho está sendo baseado.

2.1. Definição

Para conceituar essa área, a Associação Brasileira de Normas Técnicas – ABNT – define que “a segurança da informação é alcançada pela implementação de um conjunto adequado de controles, incluindo políticas, processos, procedimentos, estrutura organizacional e funções de *software* e *hardware*” (ABNT, 2013). Além disso, busca estudar, avaliar, desenvolver e melhorar técnicas responsáveis pela proteção de dados e informações geradas pelos mais diversos sistemas computacionais, dedicando-se também a análise e levantamento de ameaças reais.

O campo de segurança da informação surge a partir da mudança significativa nos valores agregados aos dados e informações, que deixaram de ser elementos sem valor e passaram a serem visualizados como recursos fundamentais e até como moeda de troca. Como define a ABNT, a informação passou a ser um ativo essencial para os negócios das mais diversas organizações, resultado da alta interconectividade que as expôs um alto risco de ameaças e vulnerabilidades (ABNT, 2005).

2.2. Princípios

Existem três princípios básicos, definidos nos padrões internacionais ISO/IEC¹ 27002:2005, que são considerados o coração da segurança da informação. Estes três conceitos formam a chamada Tríade CIA: Confidencialidade, Integridade e Disponibilidade (acrônimo do inglês *confidentiality, integrity e availability*), que aborda assim as principais convicções, estabelecendo uma base sólida e completa para a aplicação das técnicas desenvolvidas por essa área. Mesmo que a tríade esteja com seu papel bem definido, a Autenticidade e Não-Repúdio são dois de alguns outros conceitos que foram definidos para estender o domínio dos procedimentos de

¹ Comitê técnico conjunto entre a Organização Internacional para Padronizações – ISO – e a Comissão Eletrotécnica Internacional – IEC – responsáveis por desenvolver, manter e promover padrões na área de Tecnologia da Informação.

segurança. Nas subseções a seguir estão descritos os cinco conceitos já citados, seguidos pela descrição dos seus objetivos conforme Radack (2004).

2.2.1. Confidencialidade

O conceito de confidencialidade assegura que a visualização das informações seja restrita somente entre aqueles cuja a participação na troca seja autorizada. Ou seja, terceiros não são, em teoria, bem-vindos à conversação.

Esse princípio tem como objetivo preservar as restrições autorizadas sobre acesso e divulgação de informação, incluindo meios para proteger a privacidade de indivíduos e informações privadas. A divulgação não autorizada de uma informação é um exemplo da quebra de confidencialidade.

2.2.2. Integridade

A integridade garante que a informação recebida no destinatário seja a mesma enviada pelo remetente, bem como as formas de processamento sejam seguras o suficiente. Prevenir-se contra a modificação ou destruição imprópria da informação, incluindo a irretratabilidade e autenticidade dela é tido como seu principal objetivo.

Modificar ou destruir uma informação, significa que esta foi comprometida e já não deve ser utilizada, do mesmo jeito que uma alteração na forma de processamento da informação leva ao comprometimento da integridade.

2.2.3. Disponibilidade

A certificação de que as informações e os sistemas estarão sempre ativos e acessíveis sempre que solicitados é estipulada pelo princípio da disponibilidade. Esse conceito visa assim assegurar o acesso e uso rápido e confiável da informação.

As tentativas sem sucesso de acesso à uma informação ou a um sistema, cujo os mesmos encontram-se indisponíveis, causado por falhas típicas ou induzidas, são formas da qual esse fundamento não está sendo atendido, cabendo até a análise e aplicação de conceitos que auxiliam em sua manutenção, como os de tolerância à falha.

2.2.4. Autenticidade

Autenticidade é um dos conceitos, fora da tríade CIA, mais utilizado pelos diversos sistemas, pois garante que os participantes da troca de informações sejam quem dizem ser e que são capazes de serem verificados e autenticados na validação que antecede a troca. Além disso, certifica que as mais diversas fontes são realmente confiáveis.

Esse conceito é um dos mais complexos de ser implementado, pois garantir, por meio de um serviço digital, que no outro lado da linha está quem realmente diz ser não é tão simples, seriam necessários mecanismos que conseguissem assegurar de maneira rápida e sólida essa confirmação, assim como a comunicação humana faz através do olhar.

Atualmente o mecanismo que se propõe a resolver esse desafio são as Assinaturas Digitais, que, segundo Kurose e Ross (2014), devem ser verificáveis e não falsificáveis, comprovando assim que informações que possuam aquela assinatura, pertencem ao único que poderia assiná-la (não falsificável) e foi assinado, de fato, por ele (verificável).

2.2.5. Não-Repúdio ou Irretratibilidade

Já que as assinaturas digitais garantem a autenticidade de ambos os lados na troca de informações, o não-repúdio, não-repudição ou irretratibilidade possibilita que nenhuma das partes dessa transação neguem, futuramente, que fizeram parte da negociação em questão. Dessa maneira as transações ocorridas não podem ser contestadas ou negadas, deixando um rastro digital, que serviria para investigações, caso necessário.

2.3. Ameaças à segurança

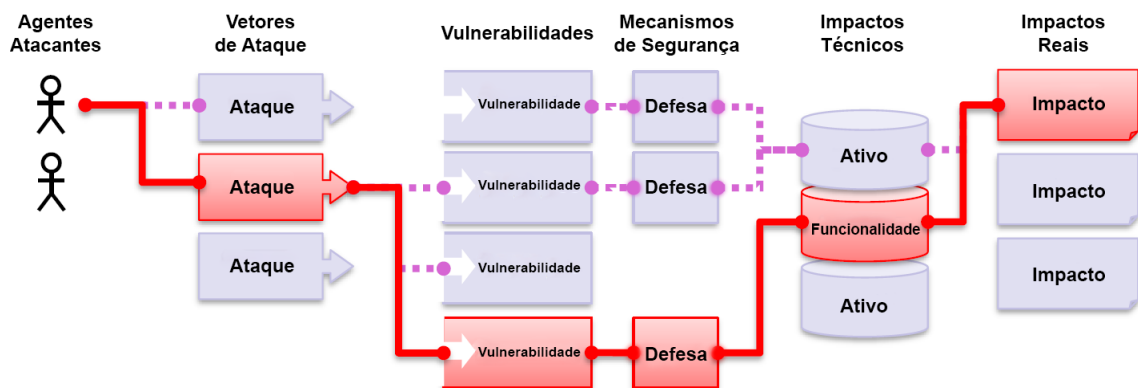
A área de SI e seus princípios existem para proteger a informação das diversas ameaças presentes, após passar por uma reformulação em termos de valores, principalmente com o advento da comunicação digital, essas informações passaram à serem visualizadas em outra perspectiva e acabaram ganhando uma exposição maior do que se esperava. As ameaças à informação são “potenciais violações de segurança, que existem quando há alguma entidade, circunstancia, capacidade, ação ou evento que possa causar prejuízo” (SHIREY, 2007), seja na aquisição e/ou

visualização indevida, na alteração, na destruição e até mesmo na não obtenção delas. As ameaças, geralmente, não existem por si só, nascendo a partir do momento em que se dá consciência de uma vulnerabilidade.

Essas vulnerabilidades “são falhas ou fraquezas no *design*, implementação, operação e/ou gerenciamento de um sistema, que podem ser exploradas a fim de violar a política de segurança do sistema” (SHIREY, 2007). Essas falhas não estão contidas nas informações em si, e sim nos sistemas responsáveis por guardá-las, sendo inevitável que a maioria deles contenha, pelo menos, uma fraqueza e isso não necessariamente o torna completamente falho.

As fragilidades de um sistema podem ser exploradas a partir de ataques, porém nem toda ameaça pode resultar em um ataque e nem todos podem suceder. Segundo a *Request For Comments – RFC – 4949* (2007), os ataques podem ser visualizados como atos intencionais da qual algum agente atacante se aproveita de alguma vulnerabilidade em um sistema, burlando os mecanismos de defesa, objetivando violar suas políticas de segurança. Porém a exploração dessas brechas depende de uma série de fatores, como por exemplo, seus níveis de gravidade, que indicam a possibilidade da obtenção de informações importantes ou não, a força ou inteligência do ataque, e a reação dos mecanismos de defesa. Tratar uma vulnerabilidade implica em custos e, geralmente, as de maior gravidade necessitam de um tratamento especial, em contrapartida, as de menor ou nenhum risco requisitam pouco ou nenhum cuidado. A Figura 1 ilustra a ocorrência de um ataque.

Figura 1 - Fluxo de ação da exploração de uma vulnerabilidade.



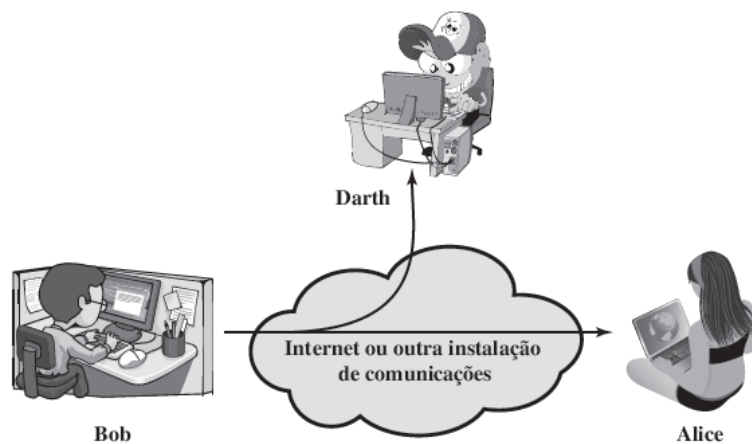
Fonte: OWASP, 2017 (Tradução Nossa).

Nota-se que os ataques existem em grande quantidade, assim como as vulnerabilidades são inúmeras, pois elas mudam de sistema para sistema. A partir disso, tanto na especificação X.800 – da *International Telecommunication Union* – ITU – como na RFC 4949, foi construída uma classificação para facilitar a identificação de ataques mais comuns, bem como as vulnerabilidades que exploradas. Os ataques estão divididos em Passivos e Ativos.

2.3.1. Ataques Passivos

Por muitas vezes, as entidades atacantes objetivam adquirir informações importantes, seja para o estudo delas e o planejamento de ataques posteriores, ou mesmo para tirar benefícios em cima dessas informações, que na maioria das vezes são privadas. Essas entidades são tidas como bisbilhoteiros ou monitores e se utilizam de ataques passivos “que, se realizados com sucesso, não resultaria na modificação de nenhuma das informações contidas no sistema, e nem o funcionamento ou o estado do sistema seria alterado” (ITU, 1991). A Figura 2 ilustra o funcionamento desse tipo de ataque:

Figura 2 - Estrutura de um ataque passivo.



Fonte: STALLINGS, 2015.

A natureza desses ataques permite que sua execução ocorra sem quaisquer problemas, pois é muito difícil detectá-lo por conta da não alteração nos sistemas e/ou informações. Com isso surge alguns padrões que tiram proveito disso, os vazamentos de dados são os ataques mais simples dessa classe. Supõe-se que um transmissor com nome Bob quer enviar uma mensagem à um receptor Alice, a mensagem é preparada e enviada, um terceiro, com intenções maliciosas e de nome Darth, captura essa mensagem e descobre o seu conteúdo. Sem os mecanismos apropriados de

defesa, Darth seria capaz de conhecer todo o conteúdo da mensagem sem muito esforço e Bob ou Alice sequer saberiam da ocorrência desse problema, pois o fluxo da troca ocorreu normalmente e nada durante esse período foi alterado, ferindo assim o princípio da confidencialidade.

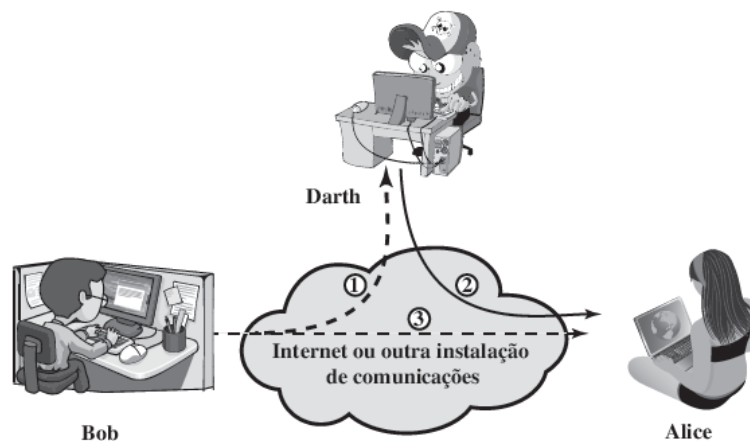
Os *sniffers* de rede (conhecidos também por analisadores de tráfego) se utilizam desse tipo de ataque para capturar o fluxo em uma rede de dispositivos. Essas ferramentas são sutis e conseguem agir de maneira a capturar os pacotes que trafegam pela rede, dando ao utilizador a possibilidade de acessar seu conteúdo. Abrindo a possibilidade de uma segunda utilização que seria, no caso, a análise fria da rede, sendo uma ferramenta de dupla utilidade.

A solução de imediato para esse tipo de ataque é o uso da criptografia, que, segundo Stallings (2015), é uma área de estudos cujo reúne métodos diversos de encriptação e de decifração, da qual, respectivamente, significa codificar o conteúdo de um texto e decodificar esse conteúdo. Utilizar esse mecanismo de defesa tornaria impossível, na teoria, para Darth ter conhecimento do conteúdo da troca de informações entre Alice e Bob. Apesar dessa solução, aplicada através da criptografia, funcionar, existem métodos de quebrar, dependendo de qual tenha sido escolhida, a encriptação utilizada, forçando a área a avançar sempre seus estudos e criar cifras melhores e mais robustas.

2.3.2. Ataques Ativos

Sendo antagonista às características dos ataques passivos, os ativos são, de certa forma, avançados. Logo que “envolvem a alteração da informação contida em um sistema, ou a mudança em seu estado ou funcionamento” (ITU,1991). Então, assim, é possível criar ataques mais complexos e que apliquem mais danos as informações ou ao sistema em si. É possível observar na Figura 3 que um terceiro componente na comunicação efetua a alteração em seu fluxo original.

Figura 3 - Estrutura de um ataque ativo



Fonte: STALLINGS, 2015.

Imagina-se um mesmo cenário da qual Bob quer se comunicar com Alice, existindo um invasor Darth. As diversas qualidades de ameaças ativas permitem, por exemplo, que Darth se passe por Alice para Bob (caminho 1), receba a informação que ele deseja transmitir, altere o conteúdo dessa informação e a repasse para Alice, passando-se agora por Bob (caminho 2), em um tipo de ataque chamado Modificação de Mensagens (do inglês, *Message Modification Attack*), quebrando assim não só a confidencialidade, como a integridade da informação.

Outra técnica de ataque, conhecida como *spoofing* é permitida através dessa classe de ataques. No *spoofing* “entidades sem autorização são capazes de ganhar acesso se passando por entidades autorizadas” (SHIREY, 2007). Isso facilita o roubo de credenciais e a quebra da autenticação, dando ao atacante a chance de ingressar no sistema e agir de maneira confortável, além de conceder o acesso as informações que deveriam estar protegidas.

Imagina-se a situação em que Darth pretende impedir que Bob troque mensagens com Alice. O atacante pode inibir a comunicação direta recebendo as mensagens que destinam Alice e as destruindo. É possível também que o atacante disfarçar-se de Alice e forjar indisponibilidade e até mesmo sobrecarregar a rede, injetando mensagens falsas na comunicação, fazendo com que as mensagens verdadeiras nunca cheguem ao destino ou que o destino esteja ocupado com mensagens falsas para visualizar e não seja possível a leitura da mensagem original. Esse procedimento chama-se Negação de Serviço (do inglês, *denial of service*), sendo muito comum sua

utilização tanto no roubo de informações (bloqueando os serviços de segurança, por exemplo) como em ataques de motivações pessoais (STALLINGS, 2015).

Embora a característica ativa desses ataques, que possibilita a identificação do autor deles, é complicado definir uma solução que os impeça de acontecer de forma absoluta. O que mais existe são saídas para a recuperação do sistema após a detecção e interrupção do ataque. Vale ressaltar que tanto nos ataques passivos como nos ativos, o autor dessa interceptação é conhecido por Homem ao Meio (do inglês, *man-in-the-middle*) (STALLINGS, 2015).

2.4. Segurança de sistemas web

O site *Hosting Facts*, em 17 de dezembro de 2018, contabilizou aproximadamente 4.1 bilhões de usuários que utilizam a internet, com uma faixa aproximada de 1.94 bilhões de websites que nela existem (*HOSTING FACTS*, 2018). Em março de 2019 o Registro.br, departamento do Núcleo de Informação e Coordenação do Ponto BR – NIC.br – responsável pelas atividades de registro e manutenção de nomes de domínio que utilizam o domínio “.br” contabilizou, no Brasil, aproximadamente 3.4 milhões de websites registrados nesse domínio (REGISTRO.BR, 2019). É possível visualizar através desses números a popularidade da utilização das tecnologias web, dentro desses diversos sites da internet estão contidos milhares ou talvez milhões de serviços operando e sendo gerenciados de forma on-line. Existem muitas empresas que administram seu negócio através de sistemas web, redes sociais que contém informações geradas a partir do próprio ingresso e criação de perfis das pessoas que a utilizam, jogos que demandam conexão com a rede, organizações que efetuam pagamentos on-line, etc.

A quantidade de informações que trafegam pela internet por dia é gigantesca, os volumes de informações importantes estão armazenados em grandes bancos de dados, atraindo a atenção de agentes mal-intencionados com relação a essas informações. Ainda segundo o site *Hosting Facts*, cerca de 90 mil websites são alvos de ataques *hackers* por dia (*HOSTING FACTS*, 2019), levantando assim a possibilidade de uma alta quantidade de informações privadas sendo obtidas e de um enorme volume de usuários sendo vítimas desses ataques.

Em 17 de janeiro 2019, aconteceu o vazamento de, aproximadamente, 773 milhões de contas de *e-mail* (G1, 2019), esse e tantos outros casos que sucederam ou não reforça a necessidade por segurança para que não haja violações de suas características (confidencialidade, integridade e disponibilidade). Atualmente existem diversos grupos e organizações voltando suas atenções para atuar nessa área, visando diminuir, o máximo possível, essas ocorrências. Geralmente, esses grupos se dispõem a estudar, desenvolver, implementar e recomendar boas práticas, recursos, ferramentas e documentações sobre segurança da informação.

2.4.1. OWASP

De acordo com Uto e Melo (2009):

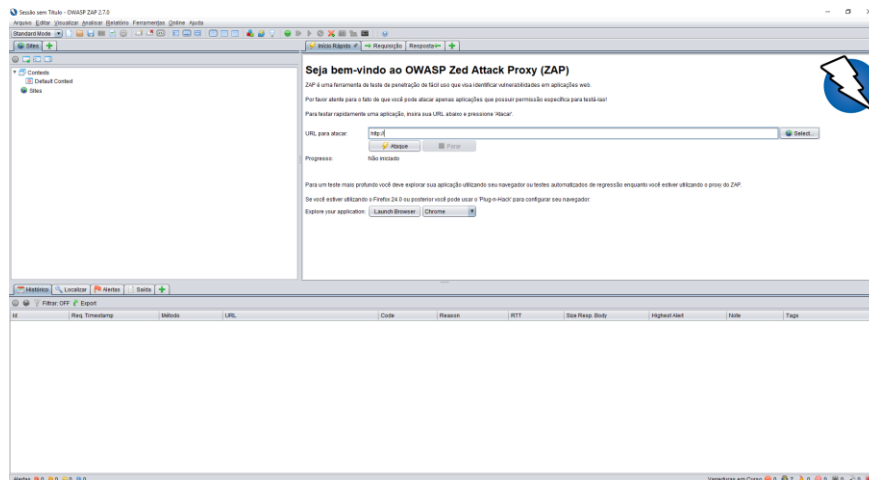
O grupo *Open Web Application Security Project* é uma organização mundial, sem fins lucrativos, que visa divulgar aspectos de segurança em aplicações web, para que o risco nesses ambientes seja avaliado por pessoas e empresas.

Essa organização surgiu no final do ano 2001, e desde então vem engajada no desenvolvimento de documentações guia para desenvolvedores *web* em todo mundo. Além de desenvolver esses guias, o grupo se propõe a realizar encontros e conferências sobre o tema ao redor do mundo, também desenvolvendo ferramentas de segurança. Seus principais projetos estão descritos abaixo:

- **Top Ten** – é uma documentação que reúne as principais ameaças a sistemas *web*, é construída através de um consenso entre esse e outros grupos, baseados em levantamentos estatísticos das últimas vulnerabilidades mais exploradas. Sua última versão foi desenvolvida em 2017 e foi o documento utilizado para a concepção de todo este trabalho.
- **Zed Attack Proxy (ZAP)** – é sua principal ferramenta para *pentesting* (explicado posteriormente) de sistemas *web*. É uma ferramenta totalmente gratuita e automatizada que faz a procura por vulnerabilidades nesses sistemas e gera um relatório completo sobre o que foi encontrado, auxiliando no desenvolvimento e teste de novos sistemas. A Figura 4 apresenta a tela inicial dessa ferramenta.
- **Testing Guide** – um guia completo com metodologias, procedimentos e ferramentas para realização de testes em sistemas *web* em desenvolvimento.

- **Code Review** – um livro em desenvolvimento que incentiva a revisão de códigos de projetos e esclarece os tipos de vulnerabilidades devem ser observadas no momento da codificação de um projeto.

Figura 4 - Tela inicial da ferramenta ZAP do grupo OWASP.



Fonte: Autoria Própria.

2.5. Pentest

Para encontrar vulnerabilidades em sistemas, ultimamente as empresas tem recorrido a metodologias que consigam, de maneira completa, não só encontrar, como explorar e avaliar essas brechas, tais testes foram batizados de *pentest* (contração do inglês, *penetration test*). Como define Weidman os “testes de invasão ou *pentesting* envolvem a simulação de ataques reais para avaliar os riscos associados a potenciais brechas de segurança” (WEIDMAN, 2014). O objetivo desses testes é descobrir o grau de ameaças que possam trazer riscos a um determinado sistema, ocorrendo em etapas e sendo exercido mais de uma vez, se necessário.

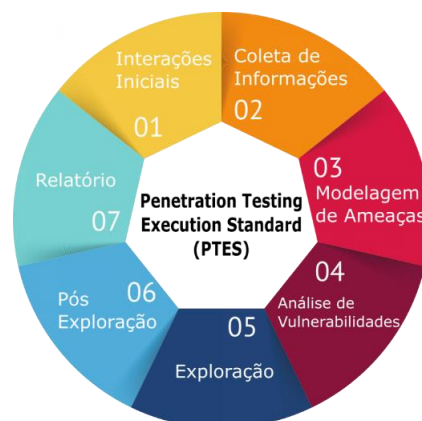
Diversas empresas surgiram focando seu ganho nesses testes, pois é cada vez mais necessário que haja profissionais nessa área para que a segurança da informação seja assegurada de maneira eficiente. Menezes, Cardoso e Rocha (2015) explicam que o profissional dessa área deve possuir um conhecimento aprofundado em programação, um domínio satisfatório das ferramentas utilizadas, uma certa visão sobre sistemas operacionais, além de conhecer, de maneira aprofundada, os conceitos de redes de dispositivos e servidores, para que dessa maneira os resultados do *pentest* sejam satisfatórios para os clientes contratantes.

Esse tipo de teste requer alguns passos fundamentais para ser executado com sucesso. Inicialmente deve haver um forte levantamento do sistema que será testado, buscas por informações relevantes, levantamento da infraestrutura sobre a qual esse sistema funciona e identificação dos serviços de rede. Essa coleta de informações dá ao profissional meios de facilitar os testes, pois aumenta a possibilidade de encontrar alguma vulnerabilidade sem necessariamente partir para a ação. Nessa fase também é definida o escopo dos testes, quais máquinas físicas serão testadas, quais testes o profissional está habilitado a executar, que serviços irão parar durante o momento de testes, etc.

Depois de levantar as informações necessárias e definir o escopo dos testes, é idealizada a estratégia de como procederá a verificação e quais métodos e ferramentas serão utilizados. Em seguida serão executados os primeiros testes para levantar as vulnerabilidades superficiais do sistema, e a partir disso é conduzida uma procura aprofundada de falhas aproximadas das já encontradas. Com os pontos frágeis levantados, é o momento em que as fraquezas são exploradas, as ferramentas são executadas e existe agora a tentativa de obter informações sigilosas do cliente, acesso as áreas restritas e controle não autorizado do sistema.

Por fim lista-se o que foi obtido, classifica-se as vulnerabilidades encontradas e o risco que elas oferecem, logo é gerado um relatório, conhecido como relatório de *pentest*, responsável por conscientizar o cliente das falhas encontradas, das informações descobertas, do que e como foi invadido e de todos os pontos cujo podem falhar e como melhorá-los. A Figura 5 ilustra como ocorre a execução desse ciclo.

Figura 5 - Ciclo de execução de um *pentest*.



Fonte: FORMIGONI, 2018.

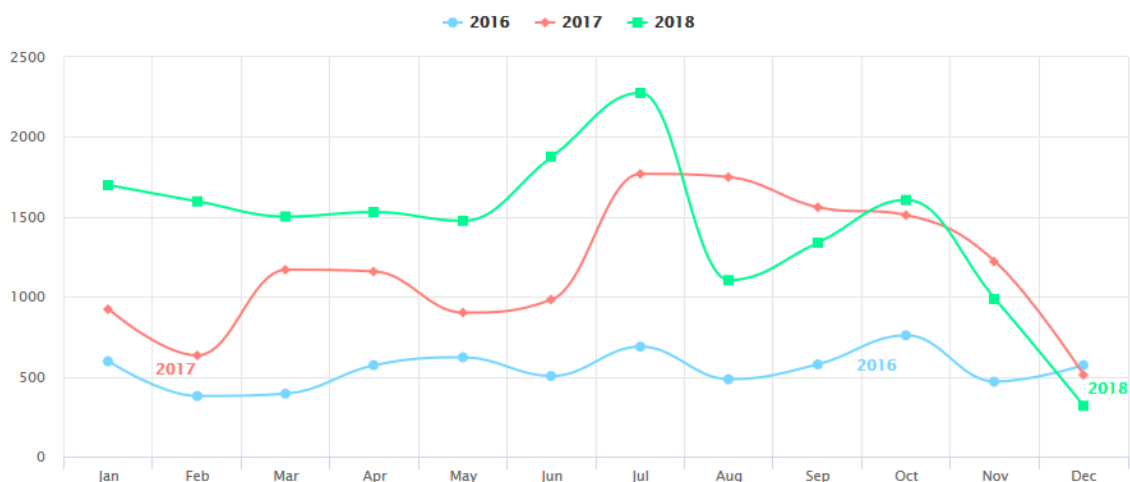
3 OWASP Top Ten

Tomado conhecimento sobre as vulnerabilidades, ameaças e ataques relacionados aos sistemas *web* e as suas informações, a OWASP decidiu classificá-las e enumerá-las para que as tornasse de conhecimento público e auxiliasse aos diversos desenvolvedores a manter seus sistemas devidamente seguros. Algumas dessas vulnerabilidades e ameaças, bem como os ataques derivados, serão abordados a seguir.

3.1. Contextualização

Devido à grande quantidade de *websites* que estão ativos na *internet*, sentiu-se a necessidade de voltar as atenções no quesito segurança dos sistemas *web*, visto que o acesso deliberado a esse tipo sistema resultou na atração de entidades mal-intencionadas, que estão sempre a empenhar-se na procura de vulnerabilidades que possam, futuramente, tornar-se uma ameaça as organizações responsáveis por manter esses sistemas. Como descreve Avital (2019), no levantamento realizado na Imperva, empresa onde trabalha, as vulnerabilidades em sistemas *web* cresceram no período de 2016 a 2018 e tornaram-se um grande problema para as diversas organizações que mantem sistemas nesse paradigma, e o que mais preocupa são as falhas graves encontradas. O Gráfico 1 apresenta o aumento no número de vulnerabilidades por sistemas *web* nesse período.

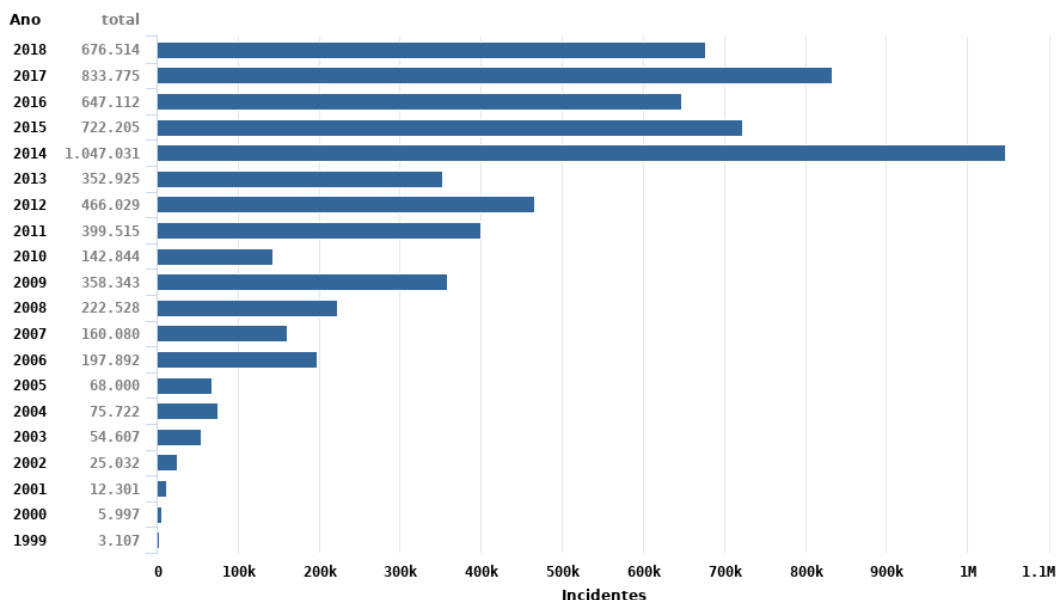
Gráfico 1 - Número de vulnerabilidades em sistemas web entre 2016 e 2018.



Fonte: AVITAL, 2019.

O Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil – CERT.br – também efetuou um estudo parecido e evidencia a quantidade de incidentes relacionados com SI. Analisando, desde 1999 até 2018, concluiu-se esses incidentes tem o propósito de causar algum dano em organizações e até mesmo em indivíduos que utilizam serviços *web* em seu cotidiano. Tendo 2014 como o ano de maior reporte de incidentes causados pela falta de segurança. Essa análise propõe que houveram dentre esses incidentes, ataques DoS, tentativas de fraudes à comércios eletrônicos e bancos, propagação de códigos maliciosos e ataques a servidores e serviços *web* (CERT.BR, 2018). O Gráfico 2 demonstra os resultados do estudo elaborado pela entidade brasileira.

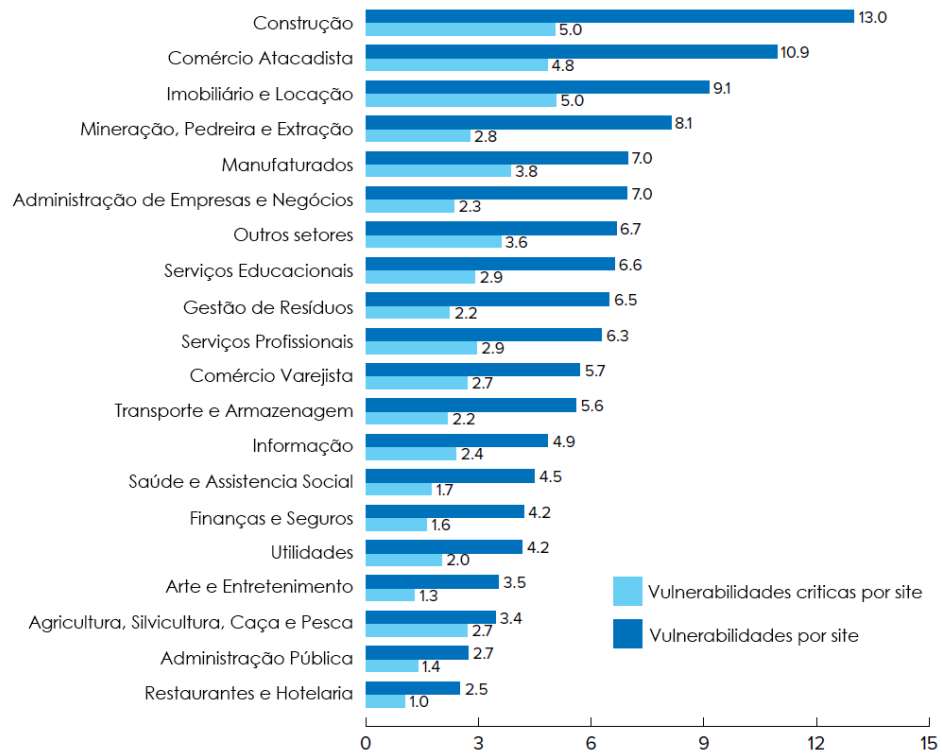
Gráfico 2 - Total de incidentes reportados ao CERT.br por ano.



Fonte: CERT.BR, 2018.

Ainda se referindo a pesquisas sobre vulnerabilidades em sistemas *web*, a *White Hat Security* (2018), em seu relatório anual, traz algumas informações importantes a respeito das vulnerabilidades encontradas em sistemas que fornecem serviços aos vários setores da indústria, reforçando assim a necessidade de melhoria no quesito segurança pelas organizações, sejam elas públicas ou privadas. O Gráfico 3 apresenta a quantidade de vulnerabilidades, bem como a quantidade de falhas críticas, encontradas nesses setores.

Gráfico 3 - Vulnerabilidades por sistema dos setores da indústria.



Fonte: Adaptado de WHITE HAT SECURITY, 2018 (Tradução Nossa).

3.2. Definição

Desde 2004, a OWASP vem fazendo o trabalho de levantar as vulnerabilidades e ameaças reais, sendo, nesse mesmo ano, liberada a primeira versão dessa iniciativa. Como descrito na seção 2.4.1 do capítulo anterior, o *Top Ten* trata-se de um relatório que expõe as principais vulnerabilidades encontradas nos sistemas *web* e visa alertar aos desenvolvedores da existência dessas falhas, que podem ser – e serão – exploradas pelas entidades mal-intencionadas, mas que são passíveis de correção. Segundo a própria OWASP (2017):

O objetivo primário do *Top Ten* é de educar os desenvolvedores, *designers*, arquitetos, gerentes e as organizações a respeito das consequências das mais comuns e importantes falhas de segurança em sistemas *web*.

Sendo assim é possível idealizar que grupo objetiva atingir uma conscientização generalizada, que haja um consenso entre os programadores e uma padronização na codificação dos sistemas desenvolvidos. A recomendação do grupo é “que, ainda na fase de desenvolvimento, as medidas de segurança sejam observadas pelos programadores das aplicações *web*” (CARVALHO, 2014).

A lista é organizada e priorizada de acordo com a prevalência, capacidade de exploração, níveis de detecção e impactos das vulnerabilidades levantadas. A priorização ocorre de maneira decrescente indo de A1 até A10, sendo possível a alteração na ordem das vulnerabilidades levantadas em edições seguintes, o surgimento de novas vulnerabilidades e a retirada de algumas delas. A versão 2017 do *Top Ten*, segundo a OWASP (2017), elenca as seguintes vulnerabilidades:

- A1 – Injeção:** Ocorre quando dados e/ou cargas maliciosos (conhecidos como *payloads*) são enviados ao sistema através das entradas padrão, enganando o interpretador fazendo com que execute comandos não autorizados. Essa falha é uma das mais comuns de serem encontradas.
- A2 – Quebra de autenticação:** A partir da implementação incorreta de funções ou subsistemas de autenticação, os atacantes podem comprometer credenciais, chaves e/ou sessões, permitindo o roubo dessas credenciais e/ou que se passe por outros usuários de maneira temporária ou indefinida.
- A3 – Exposição de dados sensíveis:** Acontece da má proteção à dados pessoais sensíveis, como dados de saúde, financeiros e de identificação pessoal. Esses dados, devido à má proteção, podem ser roubados e modificados conduzindo a possíveis fraudes e outros crimes.
- A4 – Entidades XML externas (XXE):** Sistemas legado com pouca ou má configuração de processadores XML permitem que entidades externas vinculadas ao XML possam ser processadas, abrindo a possibilidade de códigos maliciosos sejam executados de maneira natural pelo processador XML.
- A5 – Quebra do controle de acesso:** Provém das fracas restrições de acesso aos usuários do sistema, concedendo aos invasores a oportunidade de acessarem dados sigilosos, bem como utilizar funcionalidades restritas.
- A6 – Má configuração de segurança:** Essas são as vulnerabilidades rotineiras, resultado da configuração pobre dos servidores, serviços e aplicações que compõem o sistema. Configurações padrão ou incompletas de redes, protocolos, sistemas operacionais, frameworks e bibliotecas levam ao seu surgimento.
- A7 – Cross-Site Scripting (XSS):** As falhas XSS também são frequentes de ocorrerem. Surgindo a partir de sistemas que permitem que dados não

confiáveis fornecidos pelos usuários, sem a devida verificação, alterem as páginas *web* que fazem parte do sistema. Assim é possível executar *scripts* maliciosos no sistema e/ou no navegador dos usuários.

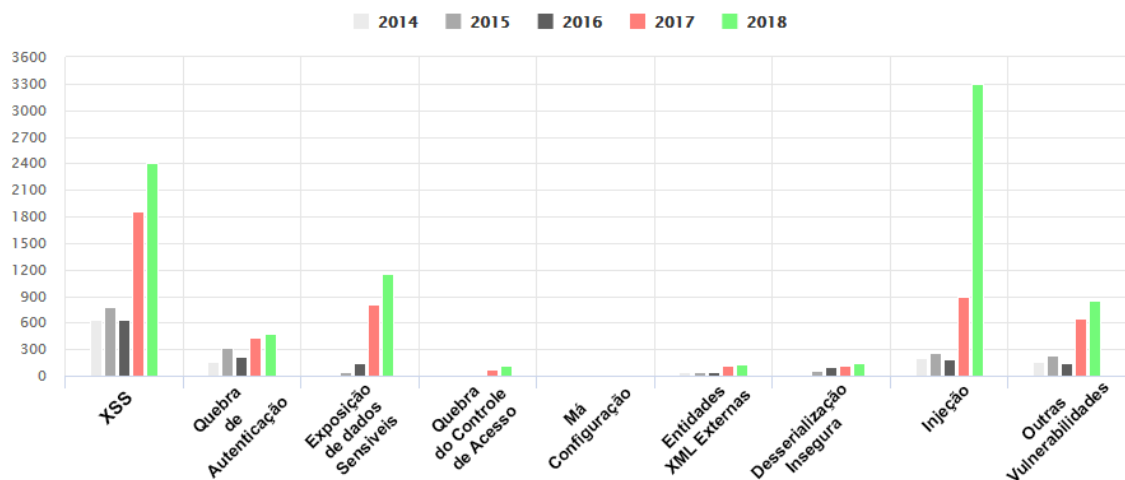
A8 – Desserialização insegura: A irregular execução de uma desserialização pode permitir que *payloads* possam ser executados remotamente, dando ao invasor a possibilidade de ganhar acesso completo ao sistema.

A9 – Utilização de componentes com vulnerabilidades conhecidas: Esse tipo de falha ocorre comumente na reutilização de componentes de *software* durante o desenvolvimento do sistema. Ao reutilizar componentes que contenham falhas já comprovadas, abre-se uma porta para ameaças iminentes ao sistema.

A10 – Monitoramento e *logging* insuficientes: A falta de monitoramento e/ou registro do que se passa no sistema, pode retirar dos mecanismos de defesa a oportunidade do pivoteamento do sistema, análise do ataque e dão ao atacante mais tempo para causar danos.

Segundo as descrições do grupo OWASP, Avital (2019) em seu levantamento, investigou e evidenciou quais dessas vulnerabilidades descritas foram as mais notadas no período de 2014 até 2018. O Gráfico 4 apresenta os resultados dessa investigação.

Gráfico 4 - Número de vulnerabilidades encontradas, entre de 2014 e 2018, seguindo a classificação da OWASP.



Fonte: AVITAL, 2019 (Tradução Nossa).

A *White Hat Security*, também utilizando como base o relatório da OWASP, realizou as seguintes observações:

1. O número de vulnerabilidades críticas crescem em uma taxa que torna quase impossível a remediação delas, se as equipes de segurança continuarem a utilizar os métodos convencionais para proteção.
2. Microsserviços (do inglês, *Microservices*) estão cheios de vulnerabilidades. Sua média de vulnerabilidades encontradas por linhas de código é maior do que a média dos serviços tradicionais, necessitando de uma correção mais rápida do que dos sistemas habituais.
3. 85% das aplicações móveis são violadas por um ou mais itens do OWASP Top Ten (versão para sistemas *mobile*).

Este trabalho se ocupa a estudar de maneira mais aprofundada dois itens dessa lista, o A1 e o A7, por se demonstrarem as vulnerabilidades mais comuns e frequentes nos diversos sistemas.

3.3. Evolução

O *Top Ten* em sua última versão sugeriu novidades em vulnerabilidades que não constavam na versão anterior a esta, datada de 2013, levando o grupo à uma completa reformulação, em termos de metodologia, processos e forma de trabalho. A OWASP se aproximou mais da comunidade e passou a dar ouvidos as necessidades que haviam. Isso se deu também graças ao avanço tecnológico e nas arquiteturas dos novos sistemas *web*. No *Top Ten 2017* a OWASP descreve:

- Microsserviços escritos utilizando Node.js e Spring Boot tomaram lugar das aplicações monolíticas usuais. Esse novo tipo de serviço veio junto com conceitos próprios de segurança, impondo à *internet* a mudança de paradigma para que se apoiassem nos serviços APIs *REST* ou *RESTful* – que são os novos paradigmas de desenvolvimento *web*, utilizando os padrões do protocolo HTTP –, que geram dados a serem consumidos por aplicações de página única, com conteúdo dinamicamente alterável – *Single Page Applications* (SPAs) – e aplicações móveis.
- Essas SPAs permitiram que os sistemas agora funcionassem de maneira altamente modular utilizando *frameworks* como *Angular* ou *React*. O que

antes era fornecido pelo servidor, que tinha seus desafios de segurança, agora é processado no lado do cliente.

- A linguagem de programação *Javascript* passou a ser ponta de lança para os sistemas *web*, sendo responsável por “dar vida” aos mais diversos *frameworks* conhecidos.

Essas observações justificam a necessidade de reformulação pela OWASP em 2017. Essas mudanças de paradigma e na forma de desenvolvimento foram cruciais para que a SI em sistemas *web* passasse a atuar de maneira diferente, adequando-se e analisando os novos conceitos e as novas formas de desenvolvimento, dando também origem à novas vulnerabilidades e, conseqüentemente, novas ameaças. Porém, em meio a toda essa evolução é possível visualizar que vulnerabilidades antigas ainda estão presentes e que não foram corrigidas em definitivo. No Anexo A é possível ver, em resumo, as mudanças entre essas duas últimas versões.

3.4. Vulnerabilidades

Como foi possível observar no Gráfico 4, as vulnerabilidades de XSS e Injeção de código (mais comumente, código SQL) foram encontradas ainda em larga escala, isso pode indicar a presença de alguns problemas como:

- Ambas vulnerabilidades não contam com soluções definitivas para as ameaças que causam.
- Os mesmos erros de desenvolvimento vêm sendo cometidos, mesmo com novas tecnologias atuando na vanguarda da *internet*.

3.4.1. Cross-Site Scripting (XSS)

Cross-Site Scripting ou XSS (o ‘X’ foi escolhido para não confundir com a sigla ‘CSS’ que significa, *Cascade Style Sheet*, conceito utilizado no desenvolvimento *web*) é uma das falhas mais frequentes em *websites*, e é comumente utilizada para fazer com que outros usuários executem código malicioso em seu navegador, geralmente esses códigos são escritos na linguagem *Javascript*. Conforme Nunan (2012, *apud* Grossman *et al*, 2007), o XSS pode ser expresso como um ataque causado por *scripts* maliciosos, da qual as entradas de usuário no sistema *web* não são devidamente verificadas, permitindo a injeção desses *scripts* e comprometendo sua integridade.

Devido ao dinamismo dado às páginas HTML com a utilização de *Javascript*, foi possível que indivíduos pudessem explorar essa falta de verificação dos dados de entrada e a partir disso conseguiam executar facilmente seus códigos maliciosos e exploravam com liberdade essa falha. Alguns danos causados às páginas *web* através desse tipo de ataque são, segundo Uto e Melo (2009, *apud* SANS, 2008):

- Sequestro de uma sessão do sistema;
- Redirecionar os usuários à outra página;
- Desfigurar as páginas HTML do sistema;
- Instalar um capturador de teclado.

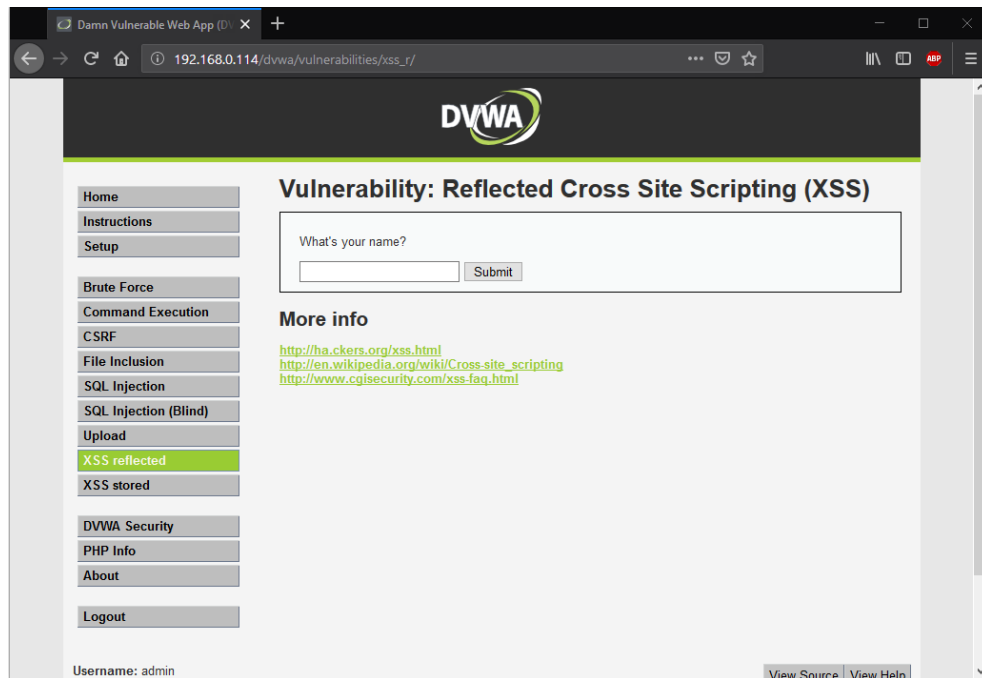
Portanto não é uma falha comum, se não houverem mecanismos de defesa eficientes pode acontecer de um usuário de alta credencial no sistema ter sua sessão tomada pelo invasor, concedendo a brecha para um comprometimento mais grave do sistema. Existem três tipos básicos de ataques XSS: Persistentes, Reflexivos e os baseados na *Document Object Model* – DOM (Grossman *et al*, 2007).

3.4.1.1. XSS Refletido

Nesta classificação de XSS, a carga maliciosa é enviada via URL ou como parte de uma requisição HTTP – *Hypertext Transfer Protocol*. Sendo assim “uma das maneiras mais comuns de envio em websites é através dos campos de pesquisa” (Grossman *et al*, 2007), que utilizam o conteúdo do campo como parâmetro para alguma pesquisa.

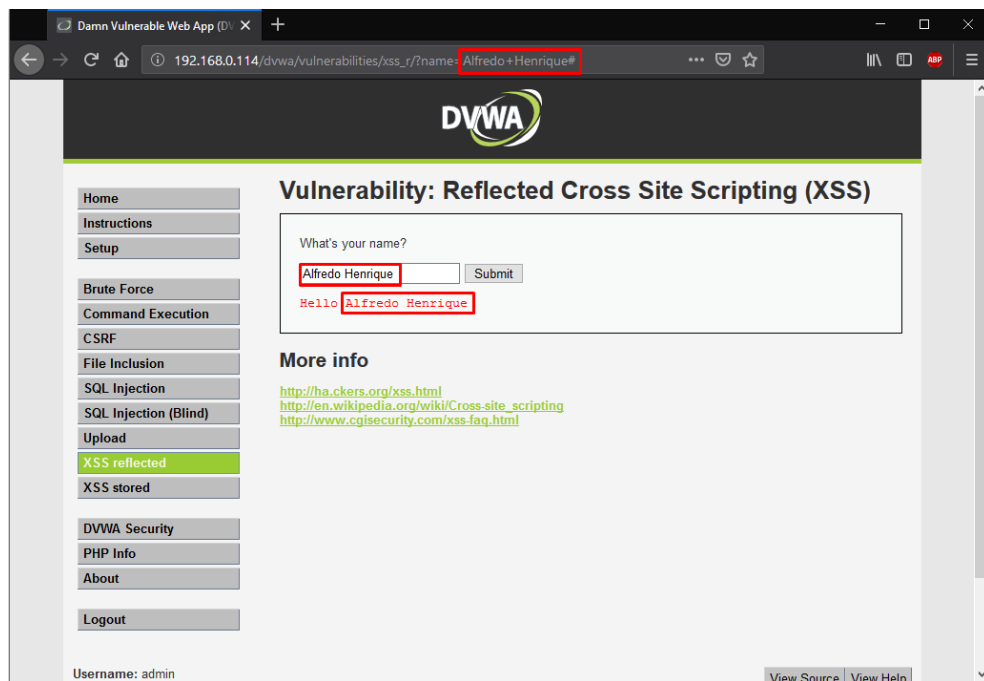
As Figuras 6 e 7 ilustram um exemplo de página *web* que contém um campo passível de ataques XSS do tipo Refletido. O valor do campo passa pela URL e dinamicamente preenche a página sem nenhum tipo de tratamento. Na Figura 8 é injetado um *script* simples, escrito em *Javascript*, que não causa nenhum mal a qualquer página ou usuário desse sistema, apenas apresenta na tela uma caixa com a frase “Permitido XSS”.

Figura 6 - Página web com campo vulnerável à XSS Refletido.



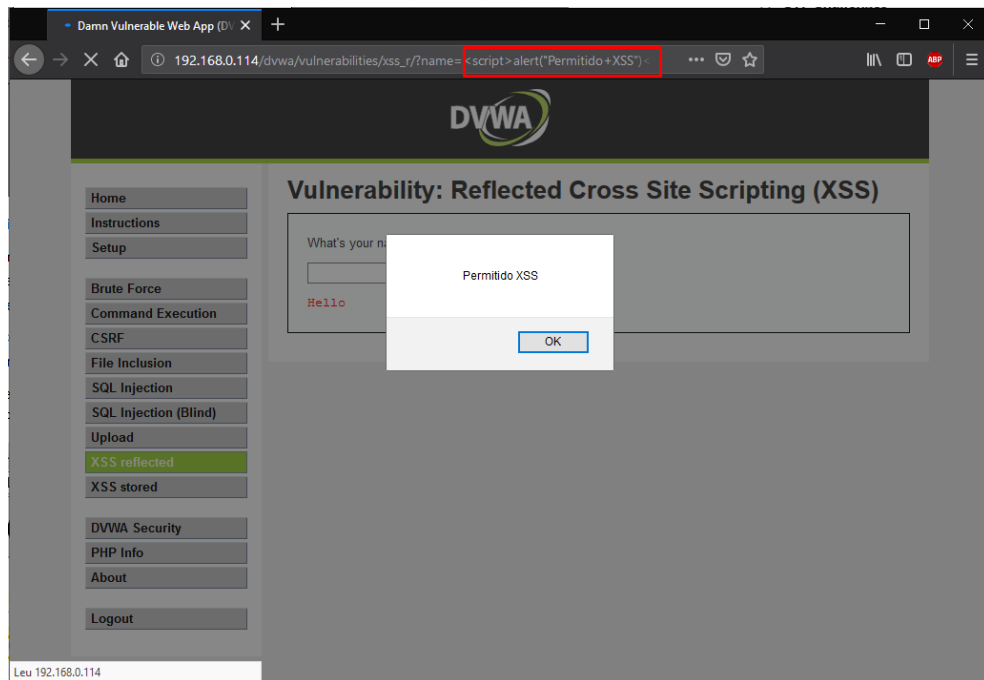
Fonte: Autoria Própria.

Figura 7 - Campo vulnerável à XSS Refletido preenchido com valores comuns e refletindo o valor sem nenhum tipo de tratamento.



Fonte: Autoria Própria.

Figura 8 - Página web após injeção de script

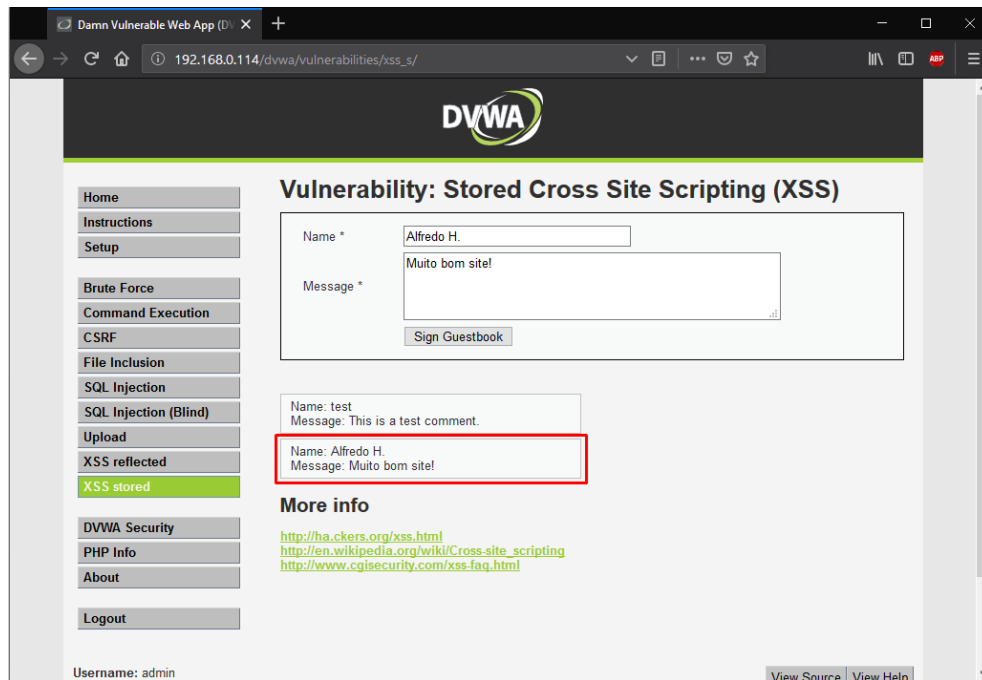


Fonte: Autoria Própria.

3.4.1.2. XSS Persistente

Os ataques persistentes (também conhecidos como injeção de HTML) “recebem esse nome porque o código malicioso é armazenado pela aplicação, normalmente em um banco de dados, e exibido a todos os usuários que acessarem o recurso” (UTO; MELO, 2009). É comum a ocorrência desse tipo de ataque em *websites* que permitem aos usuários enviar determinado conteúdo que ficará disponível para os demais usuários visualizarem, um exemplo disso, são os *blogs* que tem uma seção voltada para comentários, se não for tratada corretamente, essa seção poderá ser alvo de um ataque XSS Persistente. A Figura 9 exemplifica esse tipo de comportamento.

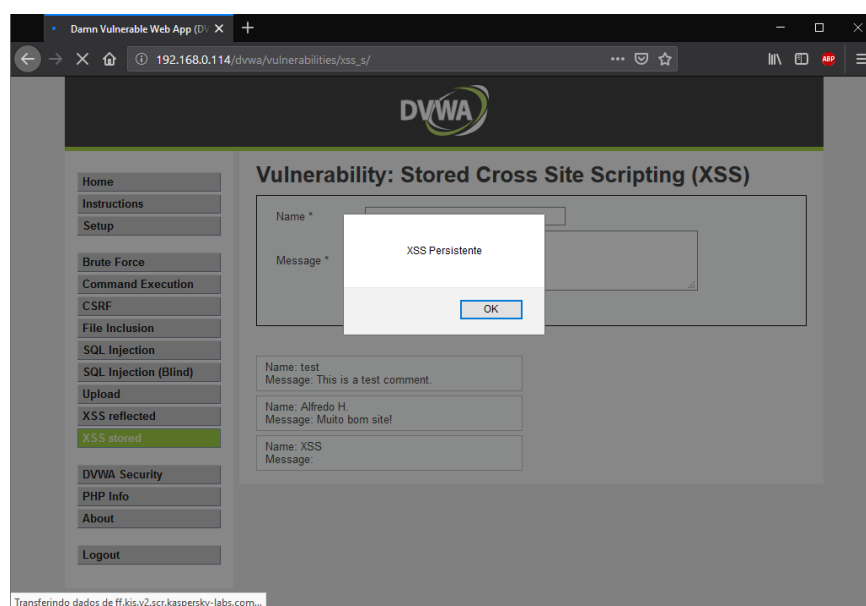
Figura 9 - Página web que permite o envio de comentários.



Fonte: Autoria Própria.

Já na Figura 10 é possível visualizar o ataque persistente sendo executado com sucesso. O *script*, novamente, não tem intenções maliciosas. Após a injeção bem-sucedida, o *script* será executado sempre que a página for acessada, oferecendo assim maior risco aos usuários, pois atinge uma quantidade maior deles.

Figura 10 - Ataque XSS Persistente executado com sucesso



Fonte: Autoria Própria.

3.4.1.3. XSS baseados na DOM

A classe de ataques XSS baseadas na DOM em muito se parece com os ataques XSS Refletidos, a alteração não afeta o servidor e ecoa apenas no navegador cliente, podendo ser definido como “o tipo de ataque em que funções *Javascript* acessam e modificam os nós e objetos da estrutura DOM, podendo referenciar, acessar ou modificar componentes do documento *web*” (NUNAN, 2012 *apud* OWASP, 2008). A alteração de textos através da URL é um bom exemplo de XSS baseados na DOM.

3.4.1.4. Contramedidas

O OWASP *Top Ten* (2017) sugere quatro medidas fundamentais para evitar a ocorrência desse tipo de ataque, como:

1. Utilizar *frameworks* que já fazem o tratamento nas entradas dos usuários nativamente, como *Ruby on Rails* e *React JS*;
2. Tratar o conteúdo das requisições HTTP para rejeitar requisições contendo *tags* HTML;
3. Aplicar codificação HTML na saída ajuda a evitar esses tipos de ataque, pois faz com que a porção de código maliciosa seja tratada como conteúdo do HTML e não parte dele;
4. Habilitar o *Content Security Police* – CSP – que é um mecanismo de defesa já implementado contra os ataques XSS.

3.4.2. Injeção de código

A injeção de código, fazendo dupla com XSS, é a vulnerabilidade que mais afeta os sistemas *web*. Essa vulnerabilidade consiste em injetar código nos formulários com campos de entrada para os usuários do sistema, e esperar o retorno de informações que deveriam ser sigilosas, ou que o sistema execute algo que não deveria executar. Apesar de ser possível injetar código em diversos subsistemas que interpretam comandos, o mais comum de acontecer é a injeção de código SQL na tentativa de obter dados armazenados no banco de dados, como por exemplo, credenciais dos usuários administradores do sistema. Para Halfond, Viegas e Orso (2006):

Aplicações web que são vulneráveis à injeção de SQL permitem ao invasor a garantia de acesso total aos bancos de dados, porque esses bancos possuem informações sensíveis sobre usuários e clientes, o que resulta em violações de segurança que incluem, roubo de identidade, perda de informações confidenciais e fraudes.

A OWASP (2017) especifica em seu relatório algumas situações que os sistemas *web* estão vulneráveis, dentre elas estão quando as entradas de usuário não são devidamente verificadas e validadas, e quando esses comandos são, através do código, usados diretamente no interpretador, facilitando assim a ação dos invasores na injeção de códigos maliciosos.

Os ataques de injeção SQL – ou SQLIA (acrônimo do inglês, *SQL Injection Attacks*) – acontecem quando consultas SQL são executadas diretamente pelos campos de formulários em páginas *web* e não são devidamente validadas, resultando no retorno de todos os dados presentes em uma tabela. A Figura 11 apresenta um exemplo de consulta construída com o conteúdo do campo de entrada. Sabe-se que um sistema está vulnerável à SQLIA quando possuir uma consulta estruturada da seguinte maneira:

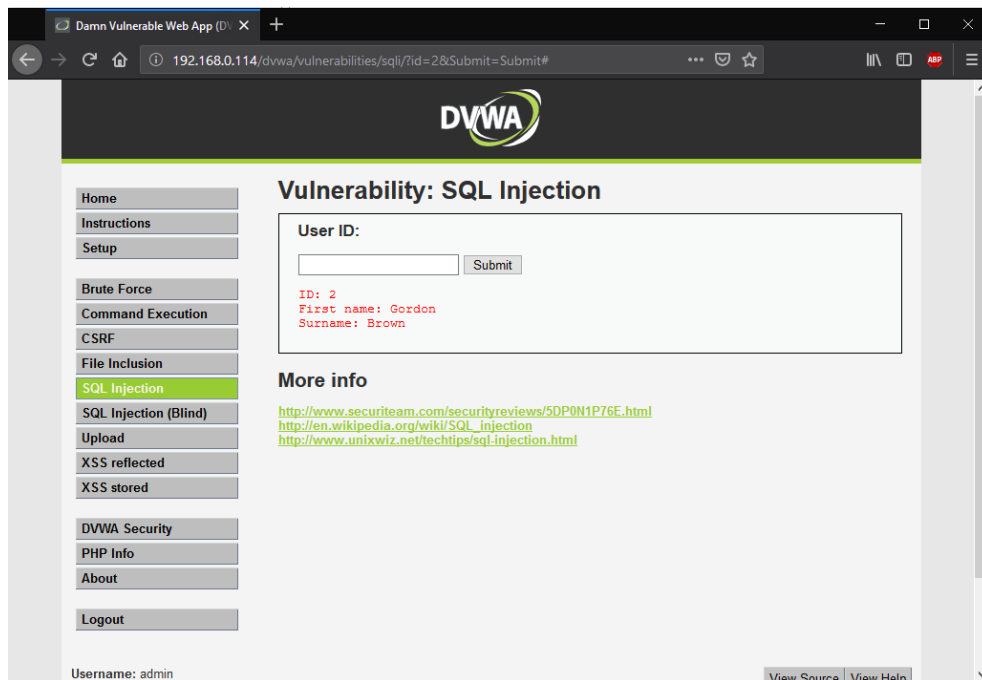
```
sqlQuery = "SELECT * FROM users WHERE  
firstName='"+inputFirstName+'"
```

Caso os usuários insiram dados comuns à entrada "*inputFirstName*", por exemplo, o sistema irá executar normalmente e irá executar a funcionalidade normalmente e retornará somente o que foi solicitado. Caso a entrada inserida seja algo do tipo " ' or '1'='1" haverá um enorme problema, logo que essa entrada resultará na seguinte consulta:

```
SELECT * FROM users WHERE firstName=' ' or '1'='1'
```

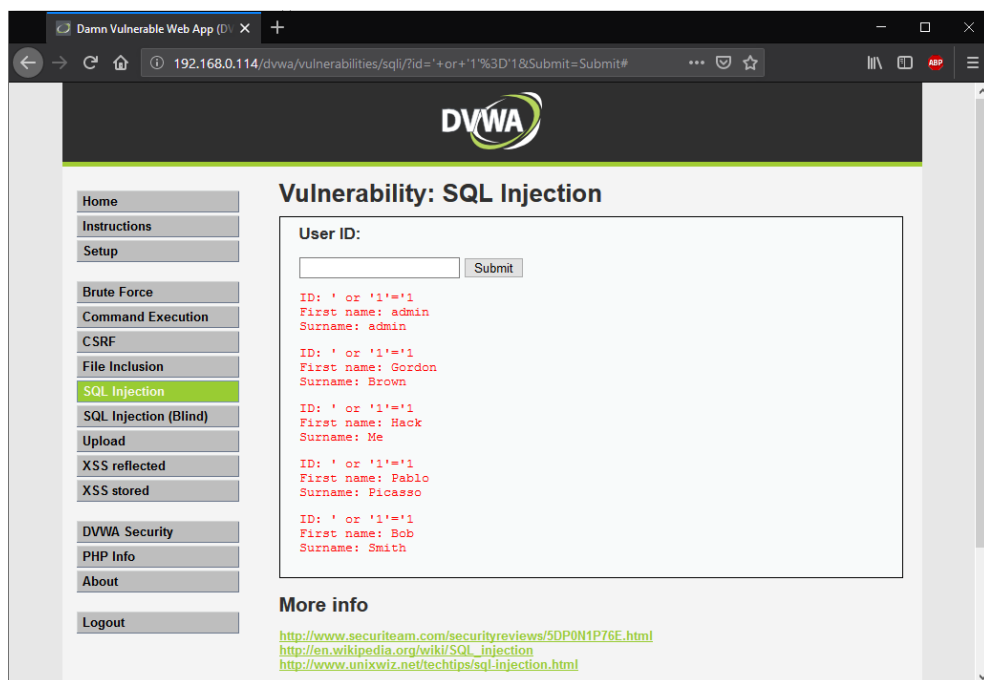
Esse tipo de consulta retornará sempre tudo que houver na tabela, logo que a cláusula *where* está sendo satisfeita pela condição "1=1", então afirma-se que essa condição é uma tautologia, já que nunca vai deixar de ser verdadeira. A Figura 12 mostra uma consulta utilizando tautologia, retornando todos os dados da tabela.

Figura 11 - Página web que utiliza o valor do ID para consultas ao banco de dados.



Fonte: Autoria Própria.

Figura 12 - Injeção de SQL do tipo tautologia no campo ID.



Fonte: Autoria Própria.

3.4.2.1. Classificação

Os SQLIA podem ser classificados quanto ao objetivo dos usuários maliciosos, dado que existem diversas consultas que podem ser executadas a partir dessa vulnerabilidade. “Para um ataque de sucesso o atacante deve concatenar um comando sintaticamente correto a consulta SQL original” (TAJPOUR; IBRAHIM; MASROM, 2011). De acordo com a categorização de Halfond, Viegas e Orso (2006), os SQLIA estão divididos em:

- **Tautologias:** Injeção de consultas com condições sempre verdadeiras, buscando o retorno de todos os dados de uma determinada tabela do banco de dados.
- **Consultas Ilegais/Semanticamente Incorretas:** Essa forma de injeção de SQL força ao banco de dados retornar um erro seguido de informações para *debugging*, que permite ao atacante levantar informações relevantes para encontrar vulnerabilidades na aplicação de banco de dados, tais como: versão, tabelas existentes, nome da *database*, etc.
- **Consultas UNION:** Esse tipo de consulta utiliza-se de um parâmetro de consulta chamado *UNION*, cujo dá a oportunidade ao atacante de concatenar a consulta, dados contidos em outras tabelas do banco de dados.
- **Consultas Piggy-Backed:** Essa técnica permite aos invasores a chance de concatenarem uma nova consulta a consulta original, utilizando o caractere de finalização de consulta “;”.
- **Stored Procedures:** Nesse caso, existem consultas pré-programadas e que estão armazenadas, apenas aguardando o momento de serem executadas. Esse tipo de consulta pode ser criado e executado posteriormente sem nenhum problema, viabilizando ao atacante novas alternativas.
- **Codificação Alternativa:** Consultas que sigam restrições de codificação podem ser burladas através de codificações alternativas, de maneira que, mesmo com as restrições, o ataque possa prosseguir.
- **Inferencia:** Muitos desenvolvedores escondem os erros gerados por consultas SQL, para dificultar os ataques de injeção de SQL, visto que os atacantes teriam uma dificuldade adicional, porém não os tornaria

impossíveis. Para isso existem duas maneiras de proceder com o ataque, de forma cega (conhecidos como *Blind Injection*), cujo o objetivo é forçar uma série de respostas *True/False* nas consultas, e de maneira a utilizar *Timing*, ou seja, injetar comandos SQL e observar a reação do sistema, mais precisamente, os *delays* nas respostas do banco de dados.

3.4.2.2. Contramedidas

Assim como para XSS, a OWASP (2017) estabelece algumas contramedidas que auxiliam no combate a esse tipo de ataque. A opção preferida pelo grupo é a utilização de APIs REST, que internamente evitam o uso de consultas diretas ao banco de dados e fazem isso através dos *Object Relational Mapping* – ORMs – que fazem o mapeamento de códigos de programação para consultas SQL de maneira segura. Outra medida é a utilização de *whitelists* que são listas contendo as consultas válidas ao banco de dados, não sendo um mecanismo de defesa completo, porém auxiliando na proteção contra algumas entradas de dados maliciosas. Por último, é recomendado o uso da cláusula *LIMIT* do SQL que restringe o número de resultados encontrados.

Em bancos de dados NoSQL, que deixaram de utilizar a linguagem procedural SQL, existe a possibilidade da injeção de cargas maliciosas em forma de *Javascript Object Notation* – JSON – artesanalmente construídos para obter dados inicialmente não disponibilizados, configurando um novo tipo de ataque nomeado como Injeção JSON.

A partir da análise dessas vulnerabilidades, como elas são construídas, como funcionam e qual o método de prevenção ao aparecimento delas, foi possível definir uma linha de desenvolvimento da ferramenta proposta por esse trabalho. A seguir será demonstrado os parâmetros, bem como os detalhes desse desenvolvimento.

4 SWAP – FERRAMENTA PARA *PENTEST* DE SISTEMAS WEB

A partir da análise e conhecimento dos conceitos apresentados nos Capítulos anteriores, observa-se a necessidade de constante verificação dos mecanismos de segurança de sistemas web e se estes podem ou não ser comprometidos em algum momento. Esse capítulo apresenta a ferramenta proposta.

4.1. A Ferramenta

Este trabalho propõe o desenvolvimento de uma ferramenta que auxilie no processo de *pentest*, automatizando-o e tornando-o mais simples de executá-lo e de obter os resultados. Os passos tomados para o desenvolvimento dessa ferramenta respeitam os passos sugeridos pela OWASP (2014), adaptando-os suficientemente para abranger o escopo trabalhado. A intenção inicial é que a ferramenta funcione no quarto e sétimo passo do ciclo de *pentest*, de maneira que possa levantar e relatar, nos sistemas web, as vulnerabilidades presentes.

A ferramenta foi desenvolvida utilizando a linguagem *Python* na versão 3.7 seguindo o paradigma de Orientação a Objetos. Essa ferramenta está dividida em quatro módulos contendo as classes desenvolvidas exclusivamente para essa ferramenta e um arquivo que contém o programa principal, que instancia essas classes. Além dos arquivos principais, foram utilizadas quatro bibliotecas, duas desenvolvidas por terceiros (*BeautifulSoup* e *Requests*) e duas que acompanham a própria linguagem *Python* (*Re* e *Urlparse*), responsáveis por dar base para o funcionamento devido da ferramenta. O nome escolhido para a ferramenta foi *Simple Web Applications Pentest* – SWAP – que referencia o objetivo principal dessa ferramenta que, além de automatizar, torna mais simples o processo de *pentest*.

O ambiente de desenvolvimento foi construído em cima do sistema operacional *Debian*, em sua versão 9.0 (*Stretch*), utilizando o *Visual Studio Code* que é uma plataforma *Microsoft* para codificação, edição e execução de novos sistemas, aplicações e/ou ferramentas. A ferramenta conta com interface gráfica em HTML, da qual recebe como parâmetro inicial o endereço do domínio que se deseja testar e os *logs* de saída são exibidos também em forma de página HTML.

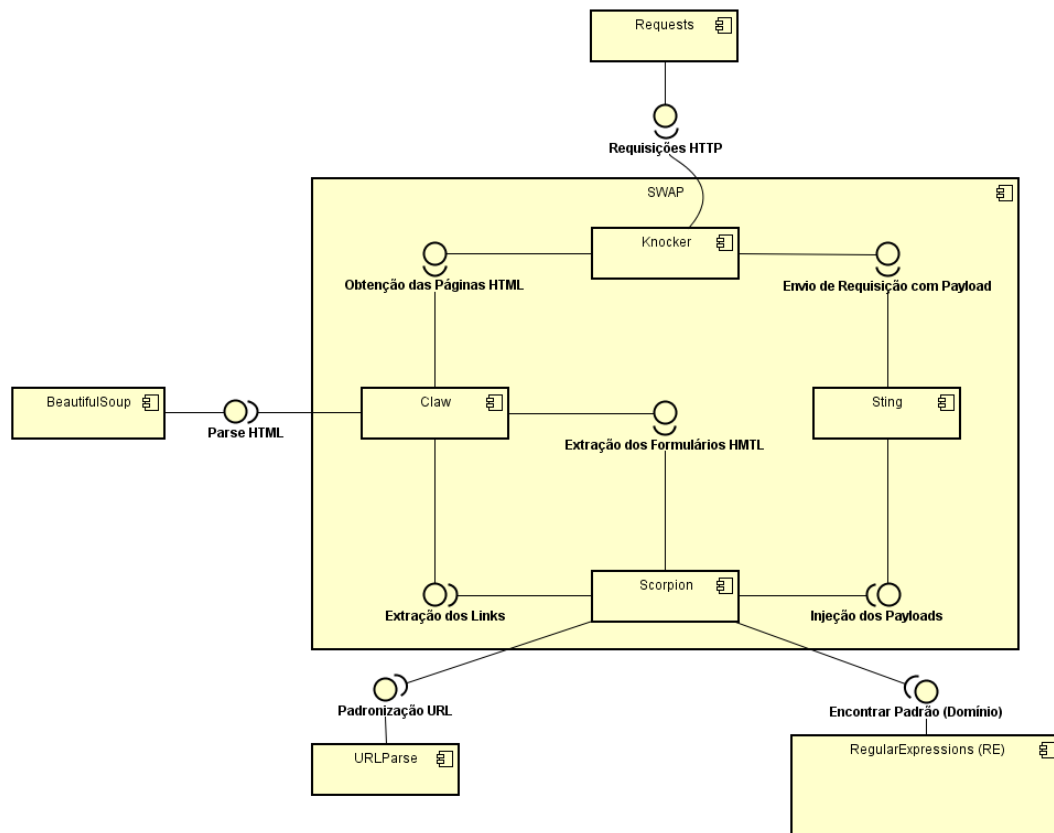
4.2. Estrutura Geral

As classes desenvolvidas para a ferramenta estão divididas entre os arquivos citados anteriormente, estando estruturadas de maneira que existe uma classe principal que utiliza as funcionalidades das demais classes. As classes estão estruturadas em seus respectivos arquivos, da qual foram nomeados com o mesmo nome da classe que armazena. As partes que compõem a ferramenta são:

- Classe *Knocker*: Fornece as funcionalidades responsáveis por efetuar requisições aos sistemas web através do protocolo HTTP.
- Classe *Claw*: Implementa um web *scraper* responsável por extrair fragmentos importantes das páginas web.
- Classe *Sting*: É responsável por injetar os *payloads* nas páginas do sistema web.
- Classe *Scorpion*: Executa o mapeamento das páginas web, extrai delas os formulários HTML, executa as injeções das cargas maliciosas no sistema e gera o *log* com os resultados das injeções.
- Programa Principal: Instancia a classe *scorpion* e executa suas funcionalidades.

A Figura 13 apresenta a estrutura dos módulos da ferramenta, bem como demonstra a relação de dependência entre as partes que a compõem através das funcionalidades fornecidas e consumidas por cada um dos módulos apresentados.

Figura 13 – Diagrama de componentes apresentando a estrutura dos módulos da ferramenta e suas dependências.



Fonte: Autoria Própria.

Os passos de funcionamento da ferramenta são demonstrados a seguir através do Algoritmo 1, bem como pela Figura 14 que os ilustra, em forma de fluxograma, demonstrando o fluxo de funcionamento da ferramenta:

Algoritmo 1 - Pseudocódigo de funcionamento da ferramenta SWAP.

Algoritmo SWAP

Início

Se dominioValido(dominio) Então

 ListaDeLinks <- extrairLinks(dominio)

 ListaDeFormularios <- extrairFormularios(ListaDeLinks)

 i <- 0

 Enquanto i < tamanho(ListaDeFormularios) Faça

 Se injetarPayload(ListaDeFormularios[i]) Então

 IncluirRegistro(ListaDeFormularios[i])

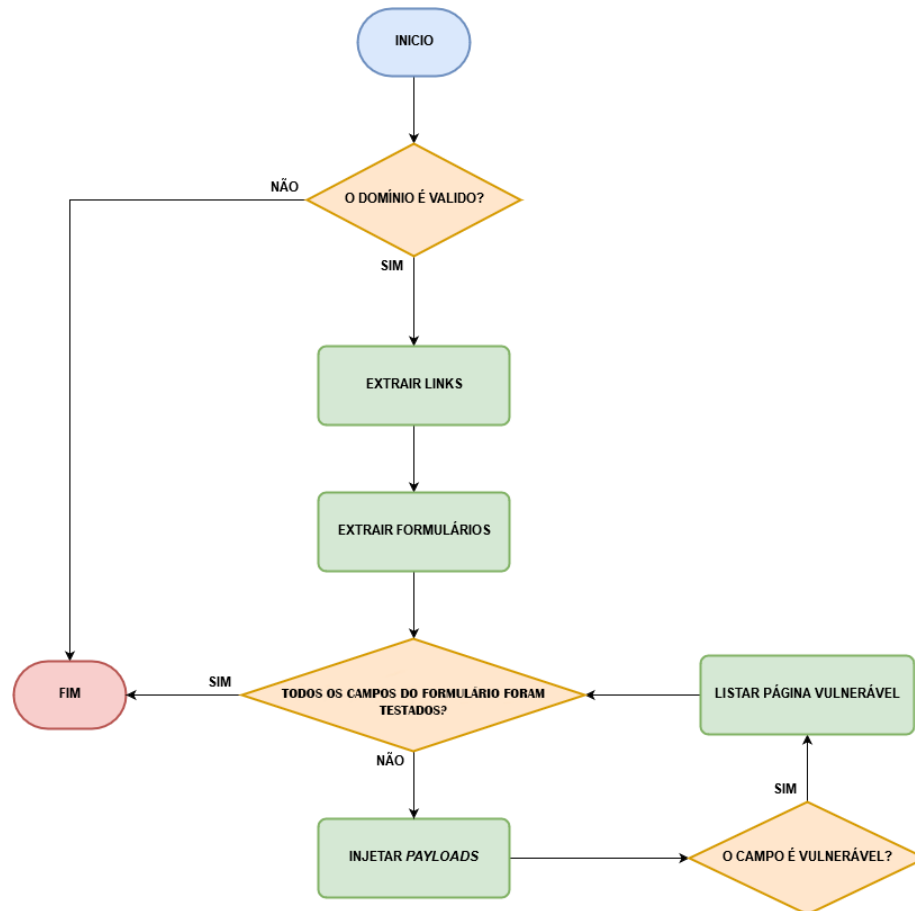
 Fim-Se

 Fim-Enquanto

Fim-Se

Fim

Figura 14 - Fluxograma de funcionamento da ferramenta SWAP.



Fonte: Autoria Própria.

Inicialmente a ferramenta recebe o endereço do sistema da qual se deseja testar. Esse endereço pode ser passado em forma de nome de domínio, como *uern.br* por exemplo, ou em forma de endereço IP, podendo ser especificado a porta e caminho do sistema, configurando uma *url* assim: *http://<nome de domínio ou ip>:<porta>/<caminho>*. Um teste é realizado para verificar o domínio através de uma requisição HTTP, caso haja resposta válida a essa requisição o sistema contido no endereço é passível de testes e a ferramenta executa as funcionalidades para o mapeamento do sistema, extração e teste dos formulários desse sistema.

Após a validação do endereço, o *web crawler* entra em ação, as funcionalidades desse módulo são utilizadas para mapear todo o sistema através da busca pelos links a partir da página inicial. Essa busca ocorre de maneira recursiva, sendo desempenhada a partir dos princípios da busca em largura nas estruturas de dados do tipo árvore. Os links encontrados são armazenados em uma estrutura de lista e

para cada novo link é verificado sua existência prévia na lista, caso exista o link é descartado, caso não exista o link é armazenado.

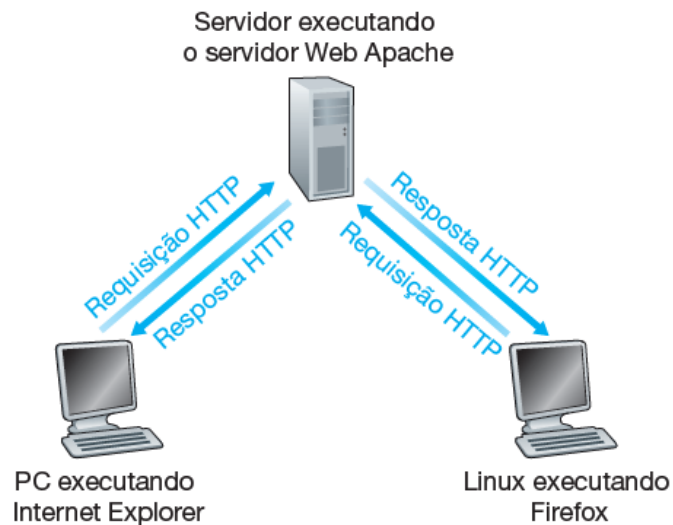
Com o sistema devidamente mapeado, o *web scrapper* visitará cada link presente na lista e extrairá, primeiramente, os formulários encontrados e em seguida os campos de entrada do tipo *text* e *textarea* são extraídos e armazenados também em uma lista. Levantando assim os possíveis pontos de entrada do sistema que posteriormente irão servir para a injeção das cargas maliciosas.

A partir desse ponto, a ferramenta injetará os *payloads* diretamente nos campos extraídos dos formulários e são armazenados em uma lista. Esses *payloads* foram artesanalmente construídos, ou seja, foram desenvolvidos a partir de testes anteriores, executados de forma manual. Dessa forma é comprovado o funcionamento desses *payloads* e que deverão retornar resultados positivos ao injetá-los. Após injetar as cargas nos campos extraídos, o conteúdo da resposta é analisado de maneira que seja avaliado a resposta recebida pela ferramenta. Essa avaliação muda conforme as vulnerabilidades testadas, podendo ser realizada em cima do cabeçalho HTTP da resposta e/ou do conteúdo da página HTML obtida. Por fim é gerado um *log* de saída em formato de página HTML contendo os links encontrados, o endereço dos formulários extraídos, a quantidade e páginas onde estão presentes as vulnerabilidades encontradas e uma descrição e classificação delas.

4.2.1. Classe *Knocker*

A classe *Knocker* é a primeira dentre as três outras classes. A escolha da nomenclatura atribuída se deu a partir dos conhecidos batedores de portas utilizados como adorno nas portas de entrada das casas, assim como item que permite ao indivíduo externo tomar conhecimento da presença de pessoas no interior das casas, bem como solicitar sua entrada. Uma alusão ao método de funcionamento do protocolo HTTP que verifica a existência de algum servidor em um determinado endereço, demanda conexão com o sistema no endereço solicitado e realiza múltiplas requisições de recursos presentes nesse sistema. É possível visualizar esse funcionamento na Figura 15.

Figura 15 - Funcionamento básico das requisições HTTP.



Fonte: KUROSE; ROSS, 2014.

O uso da biblioteca *Requests* tornou-se fundamental para a implementação dessa classe. Essa biblioteca dispõe ao desenvolvedor os métodos padrão de comunicação via protocolo HTTP: *GET*, *POST*, *PUT* e *DELETE*, em forma de funções já implementadas, do mesmo modo implementa os erros padrão caso as requisições não sucedam. Além de fornecer os métodos HTTP básicos, a biblioteca conta com outras funcionalidades que auxiliam ainda mais ao desenvolvedor na codificação de seus sistemas. Essa biblioteca conta com *features* como: Acesso ao cabeçalho e ao corpo das requisições e respostas HTTP efetuadas, gerenciamento e captura de *cookies*, verificação de certificados SSL (mecanismo de autenticação construído em cima do HTTP), autenticação básica e avançada, gerenciamento de sessão, etc.

A função da classe *Knocker* se dá a partir da necessidade de extração de elementos encontrados nas páginas web de um sistema, logo que a partir das requisições HTTP é possível obter páginas HTML completas. Alguns desses elementos a serem extraídos tem papel importante na segurança desses sistemas. Os formulários HTML fornecem aos usuários a possibilidade de efetuar entradas diretamente no sistema, sejam elas para persistir dados em um banco, para executar algum processamento e receber alguma saída, até mesmo para alterar no funcionamento do sistema. Além da necessidade de extração de elementos, a execução de testes na segurança de um sistema se dá através do envio dos *payloads* para as entradas disponíveis, ou seja, é essencial que o protocolo HTTP seja utilizado

também para transferir recursos ao sistema. No Algoritmo 2 é apresentado o código da principal funcionalidade dessa classe.

Algoritmo 2 - Funcionalidade principal da classe *Knocker*.

```
def knock(self, targetURL=None, data={}, method="get"):
    if not targetURL:
        targetURL = self.url

    try:
        if method.lower() == "post":
            return self.session.post(targetURL, data=data)

        return self.session.get(targetURL, params=data)
    except Requests.exceptions.ConnectionError:
        pass
```

4.2.2. Classe *Claw*

A implementação da classe *Claw* tornou possível a extração de elementos das páginas web. Essa classe instancia a classe *Knocker* e espera como parâmetro principal as respostas das requisições executadas por essa instância, logo que é necessário obter essas respostas, pois nelas são recebidas as páginas HTML de onde serão extraídos os elementos necessários, funcionando como um *web scrapper*, ilustrado na Figura 16. Essa função de *web scrapping* não é bem aceita na comunidade do desenvolvimento, porque essa extração dos dados contidos nas páginas de um sistema pode acontecer de forma não autorizada por seus desenvolvedores e/ou administradores, notando que essas informações extraídas podem agregar algum valor ao negócio cujo esse sistema esteja relacionado. Com isso foi concebido um novo tipo de ataque, chamado *Web Scrapping Attack*, cujo os atacantes utilizam-se de scripts automatizados que desempenham essa extração.

Figura 16 - Exemplo de funcionamento de um web scrapping.



Fonte: SAEED, 2016.

Recebidas as páginas HTML como resposta das requisições concluídas, é necessário a execução de uma análise sintática ou *parsing* para seja possível trabalhar no código HTML, de maneira a encontrar e extrair os elementos desejados. Esse *parsing* é realizado com o auxílio da biblioteca *BeautifulSoup*. A biblioteca citada não compõe as bibliotecas padrão que acompanham a linguagem *Python*, sendo responsável por transformar o código das páginas web em um conteúdo textual, do qual é possível executar algumas funcionalidades necessárias para a busca dos elementos HTML. As *tags* encontradas são retornadas em forma de lista, pois nesse caso podem existir mais de um elemento da mesma *tag*, e a partir disso, a classe monta uma lista de elementos extraídos e a disponibiliza para utilização de outras classes e/ou funcionalidades. Essa funcionalidade é executada através da função *grab* da classe *Claw* como demonstrado no Algoritmo 3.

Algoritmo 3 - Funcionalidade que obtém os elementos HTML.

```
def grab(self, htmlElement, url=None):
    if not url:
        url = self.url

    response = self.knocker.knock(url)

    if not response:
        return None

    header = response.headers

    type = header.get("Content-Type")
    if "text/html" not in type:
        return []

    parsedHTML = BeautifulSoup(response.content, "lxml")
    return parsedHTML.findAll(htmlElement)
```

A nomenclatura da classe veio a partir de uma parte anatômica de diversos animais. *Claw* – ou garra, em português – se refere a uma das funções das garras dos animais que é responsável por coletar elementos dos ambientes onde estes vivem.

4.2.3. Classe *Sting*

Sting – palavra em inglês para ferrão – remete ao membro de defesa e ataque dos escorpiões, seu ferrão. Uma alusão a injeção de códigos e scripts maliciosos nas páginas web dos sistemas testados. Essa classe é responsável por armazenar os *payloads* utilizados para testar a presença de vulnerabilidades, além disso, a classe é responsável por manipular essas cargas maliciosas em formas de requisição HTTP para que sejam injetadas no sistema e obtenha-se os resultados.

As requisições e respostas HTTP são obtidas através do uso de uma instância da classe *Knocker*, que por sua vez enviará a requisição ao sistema e obterá uma resposta. Essa resposta é analisada pela classe e é determinado que o *payload* foi processado com sucesso ou não pelo sistema, indicando o estado de vulnerabilidade do formulário encontrado na página. O resultado dessa análise é cedido como saída da funcionalidade desempenhada pela classe *Sting*. A classe conta um com dos *payloads* necessários para o teste da vulnerabilidade XSS:

```
<Script>alert(\"XSS\")</script>
```

4.2.4. Classe *Scorpion*

A classe *Scorpion* é a classe principal da ferramenta que tem a responsabilidade de instanciar as demais classes e utilizá-las. Essa classe utiliza duas estruturas de dados para armazenar dois elementos fundamentais para a ferramenta, os links existentes no sistema e os formulários HTML de entrada. Utilizando a classe *Claw* é possível extrair das páginas os links que levam para outras páginas do mesmo sistema, cada link encontrado é armazenado em uma lista de links e posteriormente é visitado para que seja executada, novamente, a extração de novos links cujo não foram ainda extraídos. Dessa maneira a classe *Scorpion* funciona como um web *crawler* que mapeia o site por completo, criando uma lista de links possíveis de serem visitados e armazena essa lista como um dos atributos da classe.

Comumente se utiliza o nome *Spider* – em português, aranha – para nomear classes e/ou módulos responsáveis pelo web *crawling*, insinuando a presença do

inseto em diversos locais e pela capacidade de se adaptarem a diversos ambientes. O nome *Scorpion* – em português, escorpião – compara o uso da ferramenta com a capacidade desses aracnídeos de também estarem em vários lugares, porém de forma mais letal pelo seu veneno, uma clara referência aos *payloads* que serão injetados nos formulários de entrada das páginas do sistema testado.

Depois de extraídos os links e mapeado todo o sistema, a classe *Scorpion*, ainda utilizando a funcionalidade da classe *Claw*, visita cada link verificando a presença de formulários HTML na página, armazenando-os quando existirem. Em seguida, os formulários são percorridos e os campos de entrada do tipo texto ou área de texto são visitados a fim de extrair os nomes dos parâmetros utilizados pelos campos nas requisições HTTP, pois são esses nomes de parâmetros que serão aceitos e utilizados pelo sistema para manipular os valores de entrada fornecidos. Essa funcionalidade é executada a partir das funções apresentadas no Algoritmo 4:

Algoritmo 4 - Funcionalidades para extração de Links e Formulários do sistema testado.

```
def extractLinks(self, url=None):
    if not url:
        url = self.url

    links = self.claw.grab("a", url)
    for link in links:
        href = link.get("href")
        href = urljoin(url, href)

        if "logout" in href:
            continue

        if "#" in href:
            href = href.split("#")[0]

        if self.domain in href and href not in self.__linkList:
            self.__linkList.append(href)
            self.extractForms(href)
            self.extractLinks(href)
    return self.__linkList

def extractForms(self, url=None):
    if not url:
        url = self.url

    forms = self.claw.grab("form", url)
```

```

for form in forms:
    formAction = urljoin(url, form.get("action"))
    formMethod = form.get("method") if form.get("method")
else "get"
    filteredInput = []
    submitName = ""

    inputList = form.findAll("input") +
form.findAll("textarea")

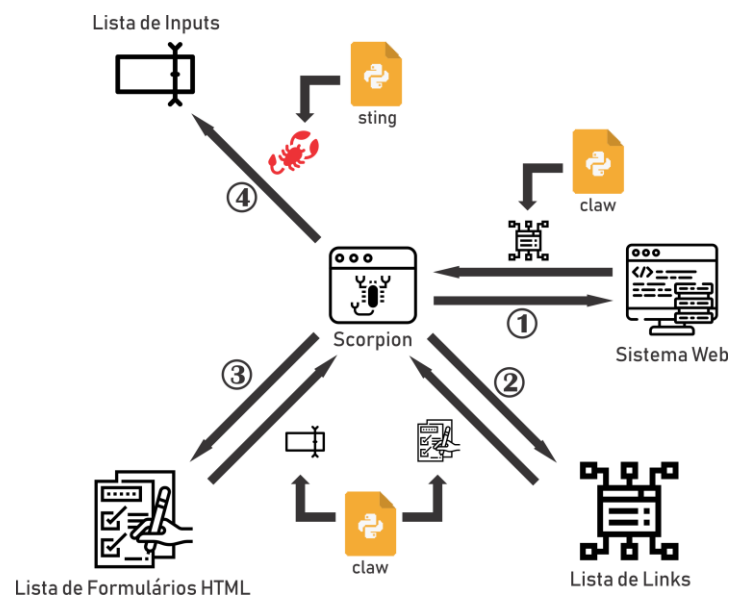
    for input in inputList:
        if input.get("type") != "submit":
            filteredInput.append(input.get("name"))
        else:
            submitName = input.get("name")

    formTuple = (url, formAction,
formMethod,filteredInput,submitName)
    self.__formList.append(formTuple)
return self.__formList

```

Com os pontos de entrada armazenados a classe *Sting* é acionada. A lista de entradas é percorrida e cada entrada armazenada é testada utilizando os *payloads* característicos para cada tipo de vulnerabilidade. Os resultados de cada teste são retornados pela classe *Sting* para a classe *Scorpion*, que os apresenta na tela para análise dos usuários dessa ferramenta. A Figura 17 ilustra os passos tomados no funcionamento da classe *Scorpion*.

Figura 17 - Funcionamento da classe Scorpion.



Fonte: Autoria Própria.

4.2.5. Interface Web

O programa principal recebe, no momento de sua execução, como parâmetro um endereço de um sistema web. Esse endereço será utilizado na instanciação da classe *Scorpion*, logo que será o endereço utilizado no funcionamento da instancia, consequentemente, da ferramenta. Depois de instanciado, o objeto da classe irá invocar os métodos responsáveis pelo funcionamento da ferramenta, respeitando os passos do algoritmo de funcionamento da ferramenta anteriormente apresentada. Esse programa é iniciado a partir da página mostrada na Figura 18.

Figura 18 - Interface inicial da ferramenta SWAP.



Digite o domínio a ser testado:

Testar

Fonte: Autoria Própria.

4.3. Trabalhos Relacionados

O desenvolvimento da ferramenta SWAP deu-se através da utilização e análise dos resultados de outras ferramentas, que se propõem a cumprir o objetivo traçado por este trabalho. Inicialmente foram testadas algumas ferramentas responsáveis pela descoberta de vulnerabilidades em sistemas web, porém essas ferramentas trabalham com o foco apenas em uma de todas as falhas conhecidas. Um exemplo disso é a ferramenta *SQLMap*.

A ferramenta *SQLMap* propõe em sua execução a descoberta, classificação e relatoria das vulnerabilidades envolvendo os bancos de dados relacionais das

aplicações web. Essa ferramenta busca expor informações sobre esses bancos, tais como: O sistema gerenciador de banco de dados utilizado, o nome do banco, a quantidade de tabelas, e, na maioria das vezes, os dados presentes nessas tabelas. Essa ferramenta cumpre seu propósito principal de maneira satisfatória, porém é uma aplicação bastante técnica, pois requer conhecimentos que, na maioria das vezes, não é familiar aos usuários estudantes, entusiastas e até mesmo leigos. Não possui interface gráfica, o que dificulta a utilização entre os usuários não-técnicos. E por fim, seu relatório é simples de ser lido, porém contém informações com alta carga técnica, que necessitam de conhecimento e estudo prévio para a compreensão correta dessas informações.

Posterior aos testes das ferramentas individuais, foram encontradas ferramentas que executassem testes para as vulnerabilidades mais conhecidas. Dentre todas as ferramentas, a OWASP Zap foi a que mais se aproximou dos objetivos deste trabalho. Seu funcionamento é semelhante ao da ferramenta SWAP, porém não conseguiu cumpri-lo de maneira integral. Essa ferramenta ao ser executada realiza o mapeamento das páginas do sistema testado, conta com interface gráfica para desktop, mostra um relatório parcial a medida em que os testes são efetuados, e, ao final demonstra um relatório de todas as vulnerabilidades encontradas, classificando-as com base no risco que essas falhas carregam.

Os problemas encontrados na utilização da OWASP Zap remetem-se, principalmente, a leitura do relatório final gerado, bem como do relatório parcial que é apresentado durante a execução dos testes. Os relatórios contêm informações de alto teor técnico e, requer dos usuários um alto conhecimento das tecnologias e paradigmas que permeiam os sistemas web, bem como o conhecimento avançado sobre as vulnerabilidades encontradas. A ferramenta é pouco personalizável, de maneira que os testes são executados detendo-se ao funcionamento previsto pelos desenvolvedores da ferramenta. Por último, foi detectado que a ferramenta não é expansível, ou seja, não permite o desenvolvimento de funcionalidades adicionais por terceiros – os famosos *plug-ins* –, exceto através da contribuição direta no código-fonte do projeto em si.

5 TESTES E RESULTADOS

Para comprovar o funcionamento da ferramenta, foram necessários sucessivos testes para análise dos resultados e conhecimento da precisão da ferramenta. Para resultados rápidos o ambiente de testes é crucial, logo que um ambiente com mecanismos bem configurados, serviços e informações bem protegidos, não é um ambiente favorável aos testes da ferramenta durante seu desenvolvimento. Esse capítulo se propõe a demonstrar os testes executados, bem como o ambiente e os resultados obtidos.

5.1. Ambiente de testes

O ambiente utilizado para os testes da ferramenta foi construído em cima de uma máquina virtual, que conta com 1GB de memória RAM, utiliza 8GB de memória secundária e manipula apenas um dos quatro núcleos disponíveis pelo processador da máquina física. O sistema operacional virtualizado é uma versão do Debian, distribuição que utiliza o *kernel* Linux, configurada para ser um ambiente de *pentest* por usuários iniciantes nessa área, esse sistema leva o nome de *Metasploitable*.

Essa máquina virtual disponibilizada pela empresa *Rapid7*, autora de uma das ferramentas de *pentest* mais famosas, a *Metasploitable*, conta com um ambiente seguro para testes de invasão de maneira controlada, pois contém diversas vulnerabilidades conhecidas disponibilizadas de maneira proposital para que seja possível o teste de ferramentas como a desenvolvida pelo presente trabalho. Essas vulnerabilidades estão contidas não só no sistema em si, mas também na forma de má configuração do próprio ambiente, em aplicações, serviços e protocolos mal configurados e/ou que contém alguma vulnerabilidade presentes em sua programação.

Essas vulnerabilidades também estão disponibilizadas em sistemas web que acompanham o ambiente virtual, aumentando a gama de possibilidade dos testes a serem executados nesse ambiente. Dentre esses sistemas disponibilizados, dois deles devem ser destacados por serem aplicações, desenvolvidas na linguagem *PHP* e que rodam em um servidor web *Apache*, que contém em sua programação diversas falhas as quais podem ser exploradas por agentes externos. O OWASP *Mutillidae* é um desses sistemas, que contém essas vulnerabilidades em sua programação de

maneira sutil. O sistema traz diversas páginas que simulam os utilitários encontrados nos diversos sistemas web desenvolvidos, transparecendo um sistema web familiar.

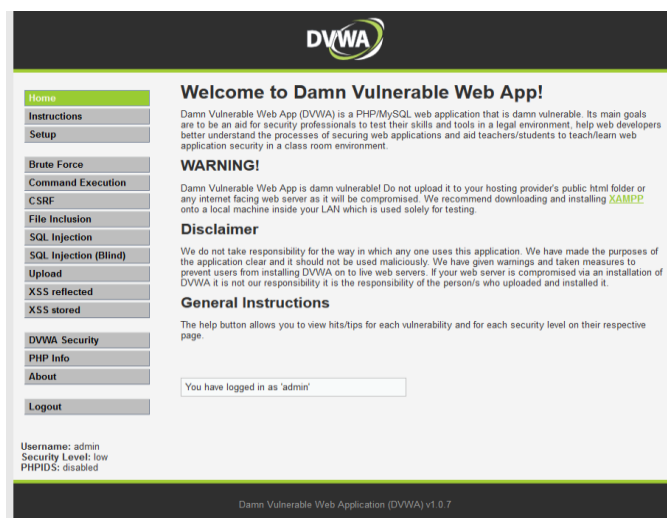
Outro sistema que fornece essas falhas é a *Damn Vulnerable Web Application – DVWA* – que traz em suas páginas, formulários bem categorizados para que possa ser realizada a descoberta de vulnerabilidades bem conhecidas. Dessa forma o ambiente transparece um sistema mais formalizado para os testes, já que as falhas disponibilizadas são explicitadas nas páginas do sistema. As Figuras 19 e 20 mostram, respectivamente, o sistema web da OWASP e o DVWA.

Figura 19 - Tela inicial do sistema *Mutillidae* da OWASP.



Fonte: Autoria Própria.

Figura 20 - Tela Inicial do sistema DVWA.



Fonte: Autoria Própria.

5.2. Metodologia dos testes

Os testes ocorreram de forma gradativa e acompanhando o desenvolvimento de cada uma das classes da ferramenta. Os testes individuais foram executados primeiramente na classe *Knocker*, a funcionalidade de requisições HTTP foi testada de maneira individual, utilizando o endereço de acesso aos sistemas web citados no tópico anterior, tendo como objetivo obter as páginas HTML iniciais utilizando o método *GET* e modificar as páginas do sistema que contém formulários HTML através do método *POST*. As classes *Claw* e *Sting* utilizam as funcionalidades da classe *Knocker*, logo foram testadas após o desenvolvimento dessa classe, sendo testadas a partir de *payloads* enviados aos formulários das páginas vulneráveis do sistema DVWA para a classe *Sting* e a partir da extração dos links da página inicial do sistema *Mutillidae* formando uma lista dos links encontrados.


Após desenvolvidas e devidamente testadas as demais classes da ferramenta, a classe *Scorpion* é testada inicialmente a funcionalidade de mapeamento do sistema, extraíndo das páginas do sistema *Mutillidae*, os links que permitem ao usuário a navegação entre essas páginas de forma recursiva. Em seguida, com os links devidamente extraídos, é testada a extração dos formulários e dos campos de entrada, formando uma lista de campos de entrada das páginas do sistema. Por último, após a extração das entradas do sistema, a funcionalidade de injeção dos *payloads* é testada, passando com parâmetro os campos de entrada armazenados, recebendo a resposta da instancia da classe *Sting* e apresentando na tela o resultado do teste.

5.3. Resultados

Por utilizar um ambiente preparado para resultados positivos na descoberta e exploração das vulnerabilidades, a ferramenta foi capaz de detectar as falhas em ambos os sistemas. No sistema DVWA a ferramenta conseguiu encontrar duas vulnerabilidades XSS nas únicas duas páginas que contém esse tipo de falha. Já no sistema *Mutillidae* foram encontradas seis páginas que continham a vulnerabilidade XSS de um total de páginas, que contém a falha, desconhecido.

Na Figura 21 é possível observar o *log* final da ferramenta, que após a extração dos links apresenta o resultado dos testes para vulnerabilidade XSS, confirmado pela Figura 22, da qual o teste para descoberta de XSS é realizado manualmente em uma das páginas vulneráveis à essa falha no sistema DVWA. É possível ter acesso ao código-fonte da ferramenta desenvolvida acessando o link a seguir: <https://github.com/AlfredoHAC/SWAP>

Figura 21 - Resultado dos testes XSS realizados pela ferramenta SWAP no sistema DVWA.

 **Resultados do Scan - SWAP**

Sumário	
Domínio	http://192.168.0.113/dvwa/
Quantidade de Vulnerabilidades Encontradas	2

Links Encontrados	
Links Encontrados	http://192.168.0.113/dvwa/ http://192.168.0.113/dvwa/setup.php http://192.168.0.113/dvwa/vulnerabilities/brute/ http://192.168.0.113/dvwa/vulnerabilities/exec/ http://192.168.0.113/dvwa/vulnerabilities/csrf/ http://192.168.0.113/dvwa/vulnerabilities/fi/?page=include.php http://192.168.0.113/dvwa/vulnerabilities/sqli/ http://192.168.0.113/dvwa/vulnerabilities/sqli_blind/ http://192.168.0.113/dvwa/vulnerabilities/upload/ http://192.168.0.113/dvwa/vulnerabilities/xss_rf/ http://192.168.0.113/dvwa/vulnerabilities/xss_s/ http://192.168.0.113/dvwa/security.php http://192.168.0.113/dvwa/phpinfo.php http://192.168.0.113/dvwa/phpinfo.php?PHPBB5F2A0-3C92-11d3-A3A9-4C7808C10000 http://192.168.0.113/dvwa/about.php http://192.168.0.113/dvwa/instructions.php?doc=PHPIDS-license http://192.168.0.113/dvwa/instructions.php?doc=readme http://192.168.0.113/dvwa/instructions.php?doc=changelog http://192.168.0.113/dvwa/instructions.php?doc=copying http://192.168.0.113/dvwa/security.php?phpids=on http://192.168.0.113/dvwa/security.php?phpids=off http://192.168.0.113/dvwa/security.php?test=%22 http://192.168.0.113/dvwa/ids_log.php

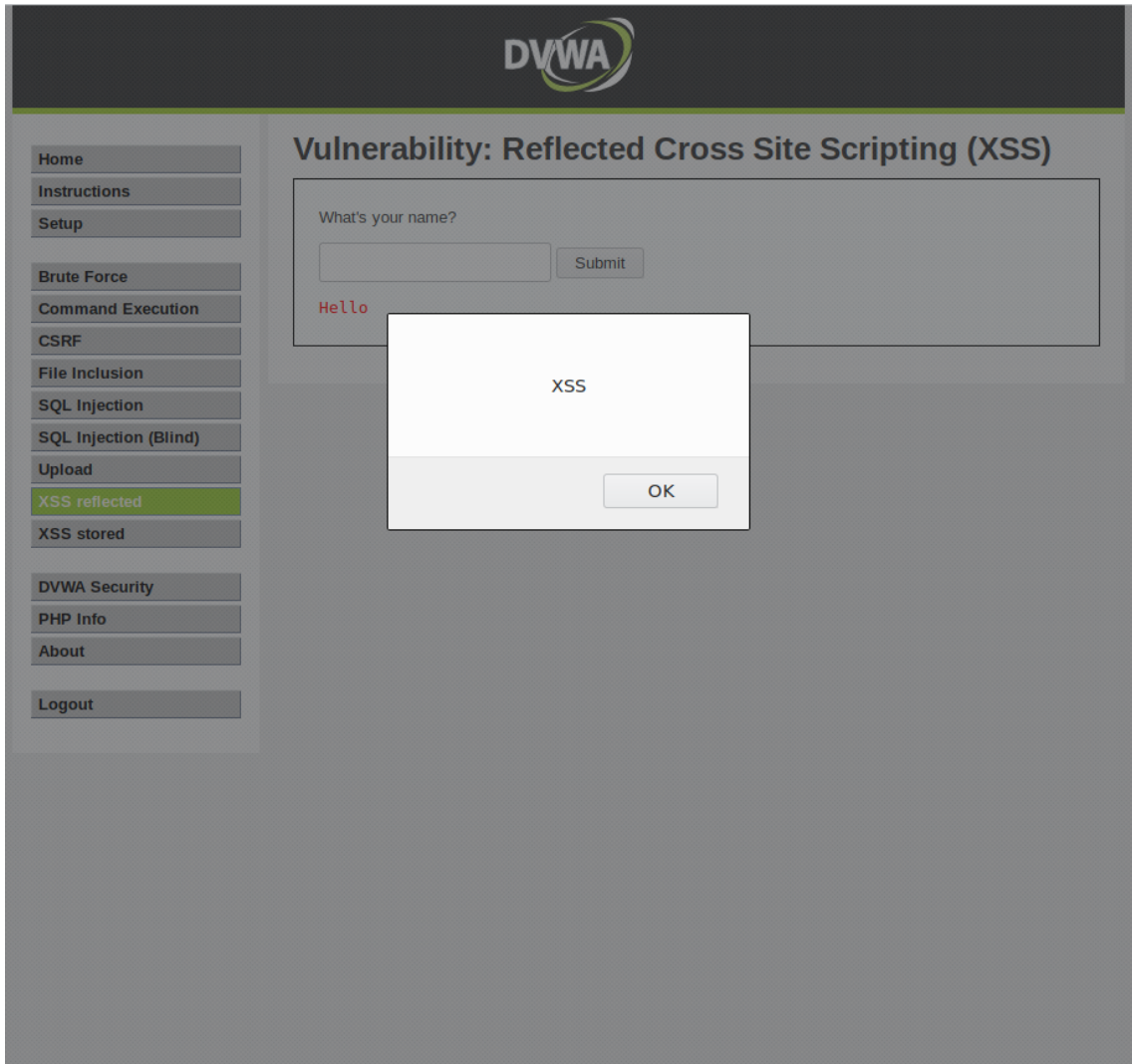
Formulários Extraídos	
Formulários Extraídos	http://192.168.0.113/dvwa/setup.php http://192.168.0.113/dvwa/vulnerabilities/exec/ http://192.168.0.113/dvwa/vulnerabilities/csrf/ http://192.168.0.113/dvwa/vulnerabilities/sqli/ http://192.168.0.113/dvwa/vulnerabilities/sqli_blind/ http://192.168.0.113/dvwa/vulnerabilities/upload/ http://192.168.0.113/dvwa/vulnerabilities/xss_rf/ http://192.168.0.113/dvwa/vulnerabilities/xss_s/ http://192.168.0.113/dvwa/security.php http://192.168.0.113/dvwa/security.php?phpids=on http://192.168.0.113/dvwa/security.php?phpids=off http://192.168.0.113/dvwa/security.php?test=%22 http://192.168.0.113/dvwa/ids_log.php

Vulnerabilidades Detectadas		
Vulnerabilidade	Formulário	
XSS	http://192.168.0.113/dvwa/vulnerabilities/xss_rf/	
XSS	http://192.168.0.113/dvwa/vulnerabilities/xss_s/	

Avançado	
Vulnerabilidade	Cross-Site Scripting (XSS)
Descrição	Ataques Cross-Site Scripting (XSS) ocorre quando um atacante injeta códigos maliciosos nas páginas de um site vulnerável a esse tipo de ataque, sendo executado apenas no Client-Side e possível enviar a página infectada para usuários comuns a fim de cometer fraudes.
Nível de ameaça	Médio
Solução	0(s) desenvolvedor(es) do sistema deve filtrar bem as entradas do sistema, de preferência, utilizar frameworks que já façam esse tipo de filtro.

Fonte: Autoria Própria.

Figura 22 – Teste XSS manual executados no sistema DVWA.



Fonte: Autoria Própria.

6 CONSIDERAÇÕES FINAIS

A proteção aos sistemas de informação é cada vez mais necessária e não se pode medir esforços para defender esses sistemas dos ataques, que estão cada vez mais complexos. Após estudo acerca dos conceitos da área de segurança da informação e a implementação parcial de uma ferramenta para *pentest* automatizado e simplificado de sistemas web, percebe-se a necessidade da constante evolução dessa área e do desenvolvimento de ferramentas que auxiliem nesse processo de levantamento.

É esperado que a ferramenta proposta contribua no processo dos testes de invasão e possibilite aos usuários economizar tempo na identificação das vulnerabilidades nos sistemas de informação, para que sejam tomadas as decisões corretas e a implementação dos mecanismos de defesa apropriados. Até mesmo que sirva para outros estudantes a voltarem suas atenções para a importância dessa área, de maneira que possam desenvolver suas ferramentas e/ou contribuir na manutenção desta.

O funcionamento geral da ferramenta ocorre de maneira satisfatória em um primeiro momento. Os resultados gerados devem ser analisados pelos responsáveis por sua utilização, logo que, encontrar vulnerabilidades não significa que estas poderão ser exploradas, bem como a não descoberta de falhas no sistema, indique que o mesmo está livre delas.

A ferramenta foi desenvolvida com limitações, mesmo funcionando como previsto. O mapeamento dos sistemas é construído de maneira rudimentar, impedindo que a ferramenta execute todos os testes apropriados. Além disso, não estão implementados os testes envolvendo todas as vulnerabilidades mais conhecidas. Exemplo disso são os testes de *SQL Injection*, que nesse caso, requerem que sejam implementados testes específicos para cada um dos tipos SQLIA conhecidos. O desenvolvimento desses testes deve ser realizado de maneira específica, de forma que todos esses testes funcionem corretamente.

Não foram executados testes em ambientes com sistemas reais, apenas em ambientes controlados, logo que, a execução da ferramenta em sistemas proprietários deve acontecer de forma legal, com o consentimento dos responsáveis por esses

sistemas. Caso ocorra sem esse consentimento, a execução dessa ferramenta é considerada uma prática ilegal, podendo se configurar crime.

A partir dessas limitações levantadas, como trabalhos futuros, estão presentes algumas sugestões para a melhoria da ferramenta:

- Execução da ferramenta utilizando ambientes reais;
- Executar testes automatizados da ferramenta, utilizando a própria linguagem *Python*;
- Melhorar a ferramenta alterando da forma de mapeamento do sistema testado;
- Limitar a execução da ferramenta em: Escopo, Tempo, etc.;
- Desenvolver um controle dos processos sendo executados;
- Execução Assíncrona e Distribuída;
- Implementação de novos testes que abordem as demais vulnerabilidades faltantes e que surgirem;
- Permitir ao usuário escolher quais vulnerabilidades deseja testar;
- Adicionar autenticação dos usuários e termos de uso;
- Disponibilizar a ferramenta em página pública com documentação apropriada.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS – ABNT. NBR ISO/IEC 27002:2005: Tecnologia da informação – Técnicas de segurança – Código para a gestão da segurança da informação. Rio de Janeiro. 2005.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS – ABNT. NBR ISO/IEC 27002:2013: Tecnologia da informação – Técnicas de segurança – Código para controle de segurança da informação. Rio de Janeiro. 2005.

AVITAL, Nadav. IMPERVA. The State of Web Application Vulnerabilities in 2018. Disponível em <<https://www.imperva.com/blog/the-state-of-web-application-vulnerabilities-in-2018/>>. Acesso em: 01/05/2019.

CARVALHO, Alan H. P. Segurança de aplicações web e os dez anos do relatório OWASP Top Ten: O que mudou?. Fasci-Tech – Periódico Eletrônico da FATEC. v. 1, n. 8, p. 6-18. São Caetano do Sul. 2014.

CERT.BR. Estatísticas dos Incidentes Reportados ao CERT.br. Disponível em <<https://www.cert.br/stats/incidentes/>>. Acesso em: 01/05/2019.

FORMIGONI, Glauber. Pentest – Passo a passo e métodos. Disponível em <<https://www.codevant.com/appsec/pentest/>>. Acesso em: 01/05/2019.

G1. Coleção de dados vazados tem 773 milhões de endereços de e-mail; descubra se o seu foi comprometido. 2019. Disponível em <<https://g1.globo.com/economia/tecnologia/noticia/2019/01/17/novo-vazamento-de-dados-tem-773-milhoes-de-enderecos-de-e-mail-descubra-se-o-seu-foi-comprometido.gh.html>>. Acesso em: 01/05/2019.

GROSSMAN, J.; HANSEN, R.; PETKOV, D. P.; RAGER, A.; FOGIE, S. Cross Site Scripting Attacks: XSS Exploits and Defense. Burlington: Syngress Publishing Inc. 2007.

HALFOND, William G. J.; VIEGAS, Jeremy; ORSO, Alessandro. A Classification of SQL Injection Attacks and Countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering – ISSSE. New York. 2006.

HOSTING FACTS. Internet Stats & Facts for 2019. 2018. Disponível em <<https://hostingfacts.com/internet-facts-stats/>>. Acesso em: 01/05/2019.

INTERNATIONAL TELECOMMUNICATIONS UNION. Security Architecture for Open Systems Intercommunication for CCITT Applications – Recommendation X.800. Geneva. 1991.

KUROSE, J. F.; ROSS, K. Redes de computadores: Uma abordagem Top-down – 6ª Ed. São Paulo: Pearson/Addison Wesley. 2013.

MENEZES, Pablo Marques; CARDOSO, Lanay M.; ROCHA, Fabio Gomes. Segurança em Redes de Computadores Uma Visão Sobre o Processo de Pentest. Interfaces Científicas – Exatas e Tecnológicas. v. 1, n. 2, p. 85-96, 2015. ISSN 2359-4942.

NUNAN, Angelo E. Detecção de Cross-Site Scripting em Páginas Web. 104 f. Dissertação (Mestrado) – Mestrado em Informática, Universidade Federal do Amazonas. 2012.

OPEN WEB APPLICATION SECURITY PROJECT – OWASP. OWASP Top Ten. 2017.

RADACK, Shirley M. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY – NIST. Federal Information Processing Standard – FIPS – 199. Gaithersburg. 2004.

REGISTRO.BR. Estatísticas. 2019. Disponível em <<https://registro.br/estatisticas.html>>. Acesso em: 01/05/2019.

SAEED, Nabeel H. Good or Evil? What Web Scrapping Bots Mean to Your Site. Disponível em <https://www.imperva.com/blog/web-scraping-bots/?utm_campaign=Incapsula-moved>. Acesso em: 01/05/2019.

SHIREY, R. INTERNET ENGINEERING TASK FORCE – IETF. Internet Security Glossary, Version 2. RFC 4949. 2007.

STALLINGS, William. Criptografia e Segurança de Redes: princípios e prática – 6ª Ed. São Paulo: Pearson Education do Brasil. 2015.

TAJPOUR, Atefeh; IBRAHIM, Suhaimi; MASROM, Maslin. SQL Injection Detection and Prevention Techniques. International Journal of Advancements in Computing Technology, v. 3, n. 7, p. 82-91. 2011.

UTO, Nelson; MELO, Sandro Pereira. Vulnerabilidades em Aplicações Web e Mecanismos de Proteção. Minicursos SBSeg 2009. IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Campinas, São Paulo, Brasil. 2009.

WEIDMAN, Georgia. Testes de Invasão – Uma introdução prática ao hacking. São Paulo: Novatec. 2014.

WHITE HAT SECURITY. 2018 Application Security Statistics Report, Vol. 13: The Evolution of the Secure Software Lifecycle. 2018.

ANEXOS

ANEXO A – Tabela de mudanças entre as duas últimas versões do OWASP Top Ten.

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]