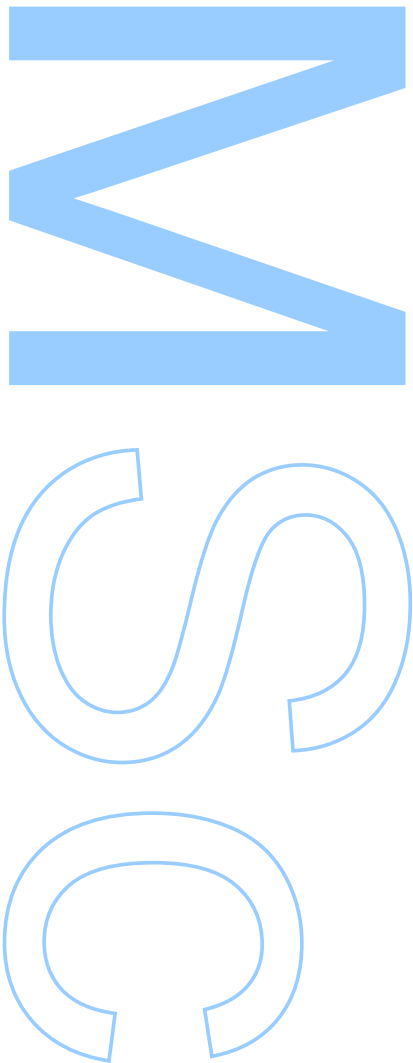
Active Vulnerability Assessment Systems

Nuno Francisco de Sá Peralta
Mestrado em Segurança Informática
Departamento de Ciência de Computadores
2021

Orientador
Luís Antunes
Professor Catedrático
Faculdade de Ciências da Universidade do Porto

Supervisor
Pedro Faria
TekPrivacy



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Abstract

Cyber threats have increased dramatically during the last decade. Malicious agents typically carry out these attacks with the intent of harming, extorting, or destroying exposed and vulnerable targets. It is vital to discover vulnerabilities more quickly and precisely in order to establish a speedier defense in terms of time in order to mitigate computer attacks. As a result, it's important to imitate an attacker's behavior in a systematic and automated method in order to find potential entry points for them, in the hopes of correcting and eliminating such attack vectors before they're exploited.

As a result, we must fix the greatest number of attack vectors in the shortest amount of time feasible, necessitating a fast and semi-automated investigation to accelerate the failure detection process. The project took form as a result of this choice. The goal of this dissertation is to find flaws with the greatest precision possible, fit them into the most diverse types of attacks currently known, and use a scalable and modular structure to allow future co-implementations and developments for large scale attacks, as well as assign a vulnerability assessment metric.

Resumo

Na última década, tem-se assistido a um elevado crescimento dos ataques informáticos. Estes ataques são normalmente efetuados por agentes maliciosos que têm o objetivo de prejudicar, extorquir ou destruir alvos expostos e vulneráveis. Para mitigar os ataques informáticos é necessário identificar as vulnerabilidades de forma mais ágil e precisa para que seja possível atingir uma defesa mais célere em termos temporais. Assim sendo, é necessário de alguma forma, emular o comportamento de um atacante de maneira sistematizada e automática para identificar possíveis pontos de entrada pelos mesmos, numa tentativa de corrigir e eliminar potenciais vetores de ataque antes que os mesmos sejam explorados.

Temos então a necessidade de corrigir o maior número de vetores de ataques, utilizando a menor janela temporal possível para o mesmo, sendo necessária uma exploração rápida e semi-automática para agilizar o processo de deteção de falhas. Foi diante esta opção que o projeto se moldou. O objetivo desta dissertação é portanto a identificação de falhas com a maior precisão possível, enquadrando-se entre os mais variados tipos de ataques conhecidos atualmente, utilizando uma estrutura escalável e modular para possibilitar futuras co-implementações e desenvolvimentos para grandes escalas de ataque e a atribuição de uma métrica de avaliação de vulnerabilidades.

Agradecimentos

Durante a elaboração deste documento, foram encontradas várias dificuldades e as mesmas foram superadas através do apoio de amigos, familiares e colegas.

É de realçar a minha mais sincera gratidão ao professor Luís Antunes pela oportunidade e pelo apoio disponibilizado ao longo deste projeto.

Também é de realçar o apoio de todos os colaboradores que se disponibilizaram a acompanhar este projeto, e pelo esforço efetuado, em particular ao Pedro Faria, Tânia Carvalho e Ana Carvalho.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Blocos de Código

Acrónimos

CVE Common Vulnerability Exposure

CWE Common Weakness Enumeration

CVSS Common Vulnerability Scoring System

HTTP Hypertext Transfer Protocol

IPS Intrusion Prevention System

SOC Security Operations Center

DoS Denial of Service

DDoS Distributed Denial of Service

SQLi Structured Query Language Injection

SQL Structured Query Language

RGPD Regulamento Geral de Protecção de Dados

Capítulo 1

Introdução

Atualmente todos sofremos de uma alta dependência tecnológica, seja a mesma proveniente de uma necessidade de comunicar com um familiar ou para efetuar um agendamento para uma vacina. Contudo, toda esta informatização e o seu consequente acesso cada vez mais acelerado revela-nos algumas preocupações. Uma delas é a existência de dados pessoais, que possuem um caráter sensível, não só a circular, mas também armazenados por parte de empresas/organizações dispersos por toda a Internet. Este acesso rápido que nos é concedido, é maioritariamente exposto também a agentes maliciosos, embora a única barreira entre os mesmos e a informação, seja uma combinação de utilizador e palavra-chave ou uma vulnerabilidade no *software*.

Os agentes maliciosos referidos anteriormente possuem técnicas e meios próprios que detêm como objetivo a manipulação de sistemas, sendo que as suas metas englobam a extração de dados relativos a pessoas singulares ou coletivas, a interrupção de serviços ou aplicações, entre outras. Estas ações podem causar enormes estragos, e até mesmo transcender do mundo virtual, tal como aconteceu no seguinte exemplo:

> Passava meia hora das quatro da tarde de sexta-feira 3 de agosto quando soou o alerta: alguns dos sistemas informáticos dos hospitais da CUF, uma das maiores redes privadas de saúde do país e que pertence ao grupo José de Mello Saúde (JMS), ficaram bloqueados. Os funcionários foram informados de que, por precaução, teriam de desligar os computadores. O azar tinha batido à porta. Os hospitais CUF eram alvo de um ataque informático protagonizado por um vírus conhecido como SamSam, que pertence à categoria dos ransomware, o que, traduzindo, significa que os dados de um computador são bloqueados e é pedido um resgate - neste caso na criptomoeda bitcoin - para os libertarem.
>
> Rui da Rocha Ferreira [49]

Devido às ameaças constantes à segurança da informação, é necessária uma revisão detalhada e concisa para identificar potenciais ataques que poderão ser efetuados ao correto funcionamento dos sistemas. É possível criar então um paralelismo como um "contra-relógio" entre uma equipa

de segurança e um atacante malicioso. Neste paralelismo, a equipa de segurança tem o objetivo de encontrar, corrigir e mitigar as vulnerabilidades enquanto que o atacante tenta identificar uma ou mais vulnerabilidades e tomar partido destas para seu próprio proveito.

Quanto ao Regulamento Geral de Protecção de Dados (RGPD) podemos afirmar que o mesmo foi tido em conta para todas as etapas necessárias na procura de uma solução viável e escalável, sendo que um dos maiores pontos a favor da sua conformidade com o mesmo é o consentimento necessário, adquirido antes de qualquer tipo de ação do programa.

1.1 Motivação

A garantia de segurança, no que toca à informática, é algo impossível de obter, dada a constante evolução, exploração e análise efetuadas por uma equipa à escala mundial na procura de vulnerabilidades [29]. Sendo que a única prática correta a adotar quanto à aproximação deste ideal, detém-se na utilização de todas as ferramentas possíveis para a mitigação de vulnerabilidades de um certo sistema, para obter o máximo de garantias e camadas de segurança possíveis. No entanto, com o objetivo de clarificar o panorama atual sobre ataques informáticos na atualidade, é possível afirmar que os mesmo sofreram um aumento de cerca de 300% [20].

Devido à pandemia de Covid-19, a vida *online* também sofreu alterações como é possível analisar no estudo de Fox [20]. Face a esta mudança no comportamento dos agente maliciosos, é possível também identificar tendências e certos padrões de ação relativos aos seus objetivos, visto que 86% dos crimes são motivados por fins monetários [59]. Segundo tais estatísticas, pode-se assumir que os alvos principais são empresas financeiramente aptas e dispostas a efetuar qualquer tipo de pagamento. Realçando também a análise realizada sobre os vetores de ataque utilizados atualmente, a figura 1.1 mostra as percentagens referentes aos ataques mais comuns.

Pela análise efetuada à figura acima referida, podemos ver que o *Phishing* [57] continua a liderar como vetor de ataque preferido pela comunidade de ciber-segurança para a execução de ataques. A mitigação do mesmo não poderá ser alcançada somente através de meios tecnológicos, e por esta razão a mitigação deste vetor será tida em consideração, mas não será um alvo prioritário do presente projeto. Foi selecionada então como meta do trabalho: a identificação e exploração automática de vulnerabilidades nos restantes casos, sempre que possível, atingindo uma análise teorética de vulnerabilidades perto dos 23% [57] de possíveis mitigações.

Para combater a tendência atual no crescimento (tanto na quantidade, como na qualidade) dos ataques informáticos são necessárias medidas de análise que propulsionem não só a eficácia na deteção de vulnerabilidades, como também medidas que tornam mais rápidas a deteção das mesmas por parte das equipas de defesa. Desta forma, é possível a preservação dos dados e dos sistemas, e portanto, são evitados os ataques referidos anteriormente.

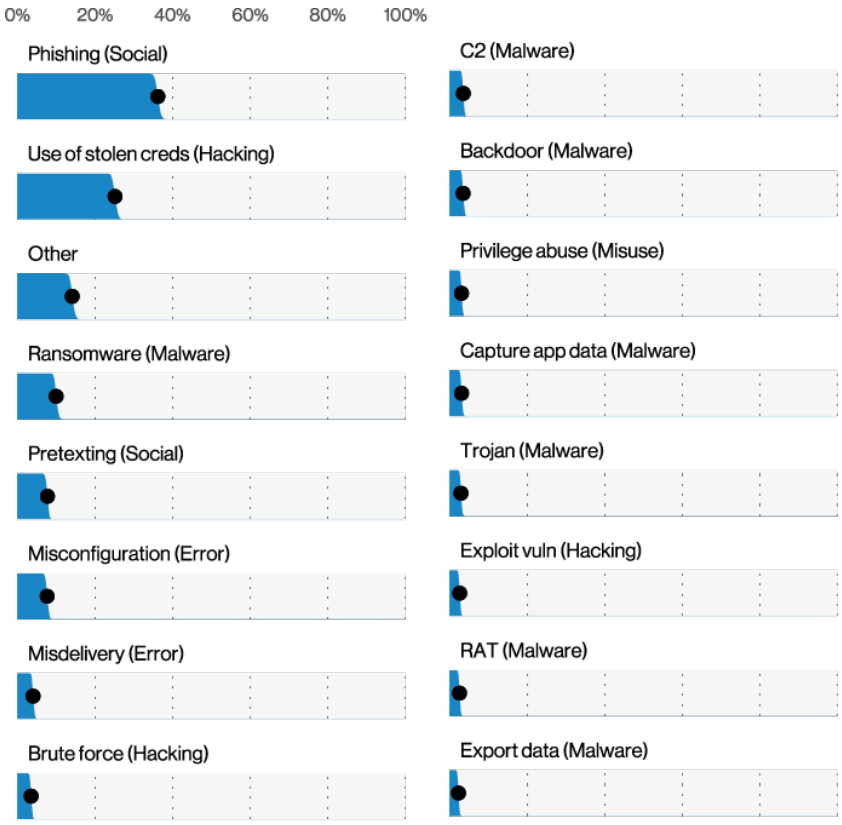


Figura 1.1: Percentagens por vetor de ataque [60].

1.2 Problema

No domínio da ciber-segurança, todos os componentes de qualquer infraestrutura que seja estabelecida como alvo para um agente malicioso, será o centro de um escrutínio detalhado que por sua vez possui apenas um objetivo: a procura do seu ”elo mais fraco” para ser quebrada toda a ”corrente tecnológica” envolvente a partir do mesmo [27].

Entenda-se então que embora um sistema, rede, infraestrutura ou qualquer componente tecnológica seja considerado seguro, de um momento para o outro pode sofrer um ataque, visto que a ciber-segurança é uma área de mutação e evolução constante. O que é considerado seguro hoje, nas próximas horas com a publicação de um novo potencial vetor de ataque, pode deixar de o ser. Estes factos estão contemplados, tendo uma denominação própria como "zero-day" *exploits* [7] que representam vulnerabilidades novas/recentes que ainda não foram expostas ao público, tornando-se assim um vetor de ataque inquestionavelmente perigoso. Esta falta de certeza leva a uma atenção constante sobre as tecnologias que um indivíduo detém e a sua comparação periódica com bases de dados sobre vulnbest ikarierabilidades, para conseguir identificar se o mesmo está (ou não) seguro. Embora o conceito de rede segura seja algo idealístico e não atingível do ponto de vista prático, temos a utilização do mesmo como um exemplo. Esta rede só precisa de ”um

elo mais fraco", como foi referido anteriormente, para ficar exposta e ser um potencial alvo de um ataque malicioso, visto que estará comprometida graças à mais pequena falha explorada por um atacante.

Em suma, para além da situação de monitorização exaustiva, podemos também concluir que é crucial uma validação contínua dos sistemas. Tal validação terá de seguir uma abordagem holística, visto que com a mais pequena falha, o prejuízo poderá manifestar-se de diversas vertentes: vertente monetária, na vertente de disponibilidade de um determinado serviço/aplicação (entre outros) ou em casos extremos, sobre a forma de confiança no alvo por parte dos indivíduos ou empresas que utilizam o seu modelo de negócio.

Estas situações derivam em parte também das exigentes regulações às quais estão sujeitas, das quais surgem a necessidade de possuir controlos para provar a sua segurança e privacidade. Como tal, é necessário o desenvolvimento de métodos que permitam a correta validação dos sistemas face à sua segurança informática. Para isso, a melhor forma de recriação destes ataques, passa por emular o comportamento de um agente malicioso de modo a providenciar uma defesa rápida e eficaz ao problema detetado.

1.3 Proposta de solução

Para combater o problema descrito anteriormente, são necessárias diversas verificações. Contudo, o método de verificação mais eficaz é a emulação de um ataque num ambiente controlado e com o devido consentimento [9]. Tal emulação é executada rigorosamente com o mesmo *"modus operandi"* do qual nos estamos a tentar defender. A solução ideal consiste na elaboração de uma plataforma com o objetivo de reduzir significativamente o número de casos falsos-positivos e falsos-negativos no momento dos testes à segurança informática de um alvo.

De modo a alcançar o objetivo acima indicado, é necessária a intervenção humana no programa, nomeadamente para a exploração de vulnerabilidades para garantir o correto funcionamento do ataque e a sua respetiva validação.

Ainda incluído no âmbito desta dissertação, propomos uma plataforma que permite a reutilização e integração dos dados necessários e produzidos pela aplicação por outras componentes que poderão ser implementadas com a intenção de providenciar a escalabilidade que a aplicação necessita para ser utilizada em grandes dimensões.

1.4 Contribuições no mundo real

O tema da presente dissertação foi proposto em parceria com a TekPrivacy, empresa que centra as suas funções no desenvolvimento de soluções tecnológicas de apoio à atividade do Encarregado de Proteção de Dados (EPD) ou Data Protection Officer (DPO).

A solução proposta permitirá que o EPD, utilizador da plataforma, tenha uma maior perceção do risco a que os dados tratados na sua organização estão sujeitos. Por conseguinte, é expectável que o EPD consiga intervir no momento em que ocorre um certo risco no sistema de informação.

Deste modo, o resultado da solução proposta será posteriormente implementado na Tek4DPO, plataforma da TekPrivacy, como uma nova funcionalidade.

1.5 Estrutura da Tese

O presente documento inclui mais cinco capítulos, que estão descritos a seguir.

Conhecimento Prévio 2 apresenta os conceitos necessários para a compreensão de termos e mecanismos presentes no decorrer do documento.

Estado da Arte 3 providencia uma revisão dos trabalhos propostos na área, numa tentativa de perceber o que já se encontra fundamentado e criado para este campo (automatização de ataques à segurança) e também as ferramentas necessárias para elaborar uma nova ferramenta, visto que as existentes não cumprem os requisitos solicitados, nem alcançam os objetivos pretendidos neste âmbito.

Desenho e o Desenvolvimento 4 expõe as soluções criadas e metodologias utilizadas para obter o correto funcionamento da infraestrutura, mantendo também a sua escalabilidade e abstração para facilitar trabalhos futuros que necessitem de ser efetuados.

Experiências e resultados 5 apresenta o produto final e os respetivos dados obtidos através da sua execução, assim como a descrição dos cenários de testes utilizados.

Conclusões 6 último e final capítulo que apresenta um sumário das motivações e principais contribuições deste trabalho. Termina com as limitações impostas pelo mundo real e direções para trabalho futuro.

Capítulo 2

Conhecimento Prévio

Este capítulo tem como função instruir o leitor e consolidar conhecimentos previamente existentes, tendo o objetivo de agilizar a compreensão do restante documento.

2.1 *Exploitation*

Um *exploit* [5] pode ser traduzido para português como uma forma de "utilizar uma determinada falha ou vulnerabilidade de um determinado componente com o objetivo de obter uma vantagem". No contexto de segurança da informação, um *exploit* consiste na utilização de uma falha no *software* ou *hardware* para obtermos uma vantagem sobre o sistema alvo. Esta ação está associada ao conceito de *bug* e de vulnerabilidade, visto ambos os conceitos serem o ponto de partida para qualquer *exploit* .

Dados todos os diferentes tipos de vulnerabilidades existentes nas tecnologias atualmente utilizadas, existem *exploits* que usam vetores de ataque para diferentes formas de prejudicar uma empresa, sejam estas monetárias ou comprometer disponibilidade [55], sendo que os vetores de ataque mais comuns são os seguintes [19]:

- Denial of Service and Distributed Denial of Service

  Ambas as ramificações deste tipo de ataque, Denial of Service (DoS) e Distributed Denial of Service (DDoS) têm como objetivo final provocar uma incapacidade temporária ou permanente no serviço ou sistema alvo de responder a pedidos legítimos [33]. Para alcançar este objetivo, a tática mais usual é esgotar uma parte dos recursos na máquina alvo, causando a falta de memória ou sobrecarregando a fila de processamento [10]. A diferença entre DoS e DDoS consiste no facto de um DoS ser um nó único a efetuar o ataque, enquanto que um DDoS consiste em vários nós (normalmente coordenados por um só atacante) a efetuar a exaustão dos recursos.

- Man-In-The-Middle Attacks

Man-In-The-Middle ou MITM [11] é o nome atribuído a um ataque que compromete dois dos parâmetros fundamentais da segurança informática, a confidencialidade e a integridade. Este ataque permite a um agente malicioso intersetar comunicações entre redes, utilizadores ou infraestruturas. O seu nome deriva do facto de o atacante ser colocado "no meio" de uma comunicação, para conseguir obter a informação pretendida. Desta forma torna-se difícil a sua deteção em tempo real, visto que este tipo de ataque não revela nenhuma anomalia para as vítimas [31]. Com as fortes práticas criptográficas atualmente, é cada vez mais difícil executar este tipo de ataques à segurança informática, no entanto, ainda não existe uma mitigação total deste vetor [4].

- Ransomware

  Um Ransonware [12] como o nome indica na sua tradução, é um "rapto" de qualquer tipo de informação, que é usado posteriormente para pedido de um resgate, normalmente pago em cripto-moedas para preservar o anonimato do atacante. Tal "rapto" ocorre através da encriptação de todos os ficheiros da vítima através de um processo criptográfico com uma chave que apenas o atacante tem conhecimento. O objetivo do pagamento do resgate é reaver a chave para posteriormente desencriptar e recuperar os ficheiros previamente corrompidos pelo agente malicioso [46]. Por norma, este tipo de ataque ocorre através da exploração de uma vulnerabilidade previamente existente nos sistemas, pela abertura de um email malicioso, ou no "*download*" de um ficheiro específico e de seguida executam código que irá despoletar o ataque, notificando o atacante com a chave da vítima.

- SQL Injection

  Structured Query Language Injection (SQLi) [13] é uma das vulnerabilidades mais comuns em plataformas *web* que dependam/utilizam um sistema de base de dados relacional. Este tipo de ataque consiste em manipular a introdução de dados com o objetivo de modificar, eliminar ou extrair informação a que o atacante não teria acesso [61]. Poderão também ser executados outros tipos de comandos sobre o sistema de base dados que impossibilitem o desempenho das suas funções, visto que o atacante tem agora controlo total sobre as ações deste sistema.

2.2 *Common Vulnerability and Exposures*

Common Vulnerability Exposure (CVE) teve o seu lançamento em Setembro de 1999 e consiste numa listagem pública de falhas sobre segurança informática [34]. Este sistema é atualmente uma referência na área da ciber-segurança, visto portar uma alta credibilidade na comunidade e ser frequentemente atualizado com todo o tipo de vulnerabilidades relativas à segurança de várias componentes essenciais à informática, como por exemplo, *hardware*, *software*, protocolos de comunicação, entre outros.

A iniciativa é operada pela "MITRE Corporation" [34] que pertence e é financiada por agentes estatais dos Estados Unidos, nomeadamente pela "US National Cyber Security Division". A

"MITRE Corporation" optou por uma padronização de todas as entradas nesta base de dados, sendo que todas as vulnerabilidades têm identificadores únicos para além de seguirem uma métrica específica para facilitar a navegação [35] como por exemplo:

- CVE - Sufixo
- 2014 - Ano
- 6271 - Identificador único

A utilização desta base de dados detém uma importância fulcral na área de investigação sobre a ciber-segurança, sendo que a mesma permite agilizar o processo de identificação de vulnerabilidades de um sistema alvo através da catalogação. As pesquisas podem ser efetuadas através de diversos tipos de dados (exemplo: números de versões, nomes da tecnologia em questão) [36] com a finalidade de facilitar a navegação por todos os registos catalogados.

Para além do identificador único, existe também uma padronização necessária para reportar uma vulnerabilidade encontrada, tendo que satisfazer vários campos que mais tarde serão tornados públicos com a devida aceitação da vulnerabilidade por parte da "MITRE Corporation", sendo estes: tipo da vulnerabilidade, vendedor em questão e produtos afetados.

Toda esta informação é supervisionada pela "MITRE Corporation" mas mantida por uma política de "comunidade aberta" em que qualquer indivíduo pode participar, desde que encontre uma vulnerabilidade válida e efetue um relatório sobre a mesma através das fontes oficiais.

2.3 Common Weakness Enumeration

Common Weakness Enumeration (CWE) [37] significa em português "enumerações de fraquezas comuns" e visa sintetizar a função da base de dados mantida pela comunidade mundial de ciber-segurança. A principal preocupação desta inicitiva consiste em classificar os vetores de ataque, assim como em casos extremos a criação automatizada de ferramentas para identificação dos mesmos [38]. Este projeto é orquestrado também pela "MITRE Corporation".

Esta iniciativa possui dados com uma enorme especificação, que por sua vez são mapeados e sub-divididos para aumentar o rigor da sua classificação e para facilitar a sua procura, sendo que os parâmetros utilizados para este fim são os seguintes:

- **A utilização de passwords padrão** em diversas aplicações que fazem com que o acesso a painéis de administração seja trivial.
- **Falta de sanitização na introdução de dados** inseridos pelo utilizador, que poderá originar diversos vetores de ataque através da execução de código ou exfiltração de dados.

Seguidos destes dois exemplos, é de ressalvar que esta base de dados não se foca nas vulnerabilidades ou falhas em si, mas em aspetos referentes à segurança informática que são agnósticas de tecnologias e podem ser explorados em grande parte da superfície tecnológica de qualquer rede, sendo a mesma pública ou privada. Na tabela 2.1 é possível verificar as dez falhas comuns que detêm a maior pontuação segundo a "MITRE Corporation" para esta classificação CWE.

Rank	ID	Name	Score
1	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
2	CWE-787	Out-of-bounds Write	46.17
3	CWE-20	Improper Input Validation	33.47
4	CWE-125	Out-of-bounds Read	26.50
5	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
6	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
7	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
8	CWE-416	Use After Free	18.87
9	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
10	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44

Tabela 2.1: Top 10 CWE MITRE [39].

2.4 Common Vulnerability Scoring System

Common Vulnerability Scoring System (CVSS) é uma *framework* gratuita que tem o objetivo de atribuir pontuações referentes à gravidade e/ou impacto de vulnerabilidades informáticas, numa tentativa de facilitar a correção e priorização de tais vulnerabilidades. Assim, é possível distinguir vulnerabilidades críticas de vulnerabilidades com um menor impacto e/ou repercussões [42].

As pontuações são calculadas através de métricas específicas sobre o impacto e facilidade de utilização de um determinado *exploit*, ou seja a complexidade e o impacto são componentes de extrema importância para esta classificação. Estas pontuações são atribuídas numa escala que varia entre 0 e 10, sendo 10 uma vulnerabilidade crítica. Podem-se observar os diferentes grupos de métricas, utilizados para calcular as respetivas pontuações através da tabela 2.2.

O grupo de métricas base da classificação CVSS é constituído por dois elementos, sendo estes as métricas sobre o *exploit* e sobre o seu impacto. É de ressalvar que todos os campos detidos por estes parâmetros são universais a todos os sistemas, e podem ser aplicados a qualquer vetor de ataque, independentemente do fator temporal ou ambiental do mesmo.

As métricas do exploit, focam-se no funcionamento do ataque utilizado, identificando os pontos de partidas e condições necessárias à sua execução, identificam também critérios externos ao controlo do agente malicioso para o alcance do vetor de ataque completo (como por exemplo: outro utilizador da plataforma ter de clicar num *link* malicioso).

<table>
<tr><th colspan="2">Grupo de Métricas Base</th><th>Grupo de Métricas Temporais</th><th>Grupo de Métricas Ambientais</th></tr>
<tr><td>Métricas do Exploit</td><td>Métricas de Impacto</td><td>Maturidade do código do exploit</td><td>Métricas Base Modificadas</td></tr>
<tr><td>Vetor de Ataque</td><td>Impacto sobre a confidencialidade</td><td>Nível de resolução</td><td>Requisitos de Confidencialidade</td></tr>
<tr><td>Complexidade do Ataque</td><td>Impacto sobre a integridade</td><td>Confidencialidade da fonte de informação</td><td rowspan="2">Requisitos de Integridade</td></tr>
<tr><td>Privilégios Necessários</td><td>Impacto sobre a disponibilidade</td><td rowspan="3"></td></tr>
<tr><td>Interação do/com o utilizador</td><td></td><td rowspan="2">Requisitos de Disponibilidade</td></tr>
<tr><td colspan="2">Scope</td></tr>
</table>

Tabela 2.2: Métricas relativas ao CVSS [17].

No campo sobre métricas de impacto, são identificados os 3 conceitos chave da segurança informática, que por sua vez, quando comprometidos ou quebrados através de uma ação voluntária de um agente, afirma-se que se trata de um ataque informático. Os conceitos podem ser descritos como:

- Confidencialidade: garante que as informações críticas e de caráter sensível de uma organização não podem ser visualizadas ou roubadas.
- Integridade: tem como objetivo evitar a alteração de qualquer informação, quer a mesma esteja em circulação ou persistida num determinado sistema.
- Disponibilidade: está relacionada com a acessibilidade a um determinado serviço, sendo que o mesmo não pode ser interrompido ou inutilizado, tendo a obrigação de estar disponível a qualquer momento para os utilizadores em questão.

Quanto ao *scope*, um atributo partilhado por ambas as métricas de *exploit* e de impacto, temos que o mesmo é a área relevante para um determinado caso de estudo, ou seja, a tecnologia base que se encontra sobre análise.

A vertente de métricas temporais tem por sua vez uma análise sobre a vulnerabilidade numa questão temporal, com critérios como a dificuldade em corrigir funcionamentos incorretos por parte de uma aplicação ou sistema e também analisar a eventual possibilidade de existir um futuro *patch* para o mesmo.

Quanto ao último grupo sobre as vertentes ambientais, foca-se sobre todos os fatores envolventes de uma determinada debilidade, analisando o sistema operativo e a rede na qual está inserido, ou seja, analisa uma série de fatores diretamente relacionados e/ou adjacentes com a vulnerabilidade em questão para determinar possíveis correlações entre integrações externas e a vulnerabilidade em questão.

Todos estes parâmetros possuem uma caracterização com três possíveis valores, representando cada um dos mesmos pontuações finais diferentes para o cálculo final. Após os cálculos

mencionados, existem quatro categorias para a vulnerabilidade, sendo divididas por intervalos consoante a sua pontuação final, como representado na tabela abaixo 2.3:

Severity	**Base Score Range**
None	**0.0**
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Tabela 2.3: Tabela sobre a pontuação CVSS e a sua distribuição [43].

A título de demonstração, foi selecionado o CVE-2014-7187 e a sua respetiva classificação CVSS, que pode ser consultada na tabela 2.4:

Pontuação Base	9.8 CRÍTICA
Vetor	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Pontuação de Impacto	5.9
Pontuação de *Exploitability*	3.9
Vetor de ataque (AV)	Rede
Complexidade do ataque (AC)	Baixa
Privilégios Necessários (PR)	Nenhuma
Interação do/com o utilizador (UI)	Nenhuma
Scope (S)	Inalterado
Confidencialidade (C)	Alta
Integridade (I)	Alta
Disponibilidade (A)	Alta

Tabela 2.4: Tabela de classificação CVSS sobre o CVE-2014-7187.

Apenas por uma constatação da pontuação base de 9.8, que por sua vez é a soma das duas pontuações previamente analisadas, é possível declarar que se trata de uma vulnerabilidade com um elevado impacto no sistema alvo pela elevada pontuação obtida.

2.5 Automatismo Total

O automatismo total de qualquer função ou tarefa consiste na minimização ou eliminação do fator humano substituindo-o por um sistema (máquina ou *software*) no desempenho das suas funções [8]. As vantagens do automatismo total são notórias, e podemos destacar entre elas a principal como sendo: o aumento nos fatores de produção. Esta vantagem traduz-se numa maior exploração de vetores de ataque, visto que um agente malicioso humano está diretamente dependente das suas limitações físicas e intelectuais. Por outro lado, um agente malicioso automatizado irá tentar todas as combinações possíveis durante o tempo necessário até à conclusão das suas tarefas.

Um dos benefícios implícitos seria também a libertação de recursos humanos presentes na empresa que se encontravam anteriormente no setor a ser automatizado, e o seu futuro aproveitamento para verificações manuais de situações de caráter crítico e não contemplados nos objetivos do sistema de automatização.

2.6 Automatismo Parcial

O automatismo parcial remete-nos para a utilização da *IDA-S (Information, Decision, Action and Supervision)* [15], uma *framework* que visa suportar a divisão e alocação de funções entre um sistema e um humano. A divisão de funções deverá ser em função do objetivo final, se for pretendido um resultado mais preciso e confiável, a exploração de vulnerabilidade e desenvolvimento de ferramentas adicionais deverá ser inteiramente da responsabilidade de um humano, dada a sua capacidade de adaptação aos problemas e ao facto de conseguir criar novas ferramentas ou *scripts* que possibilitam o alcance da *exploitation*. Um exemplo de um automatismo parcial utilizado hoje em dia, é o uso de um telemóvel, onde o processo de efetuar uma chamada é inteiramente automático, no entanto é necessária a introdução do número do destinatário por um interveniente humano.

Dye [16] sugere uma divisão em cinco partes da automatização parcial que incluem:

Proposta, consiste no desenvolvimento de funções para atingir o objetivo final.

Avaliação de uma ou mais propostas, tendo em conta circunstâncias específicas, podendo ser aprovadas ou rejeitadas.

Modificar a proposta inicial, que poderá ser alterada pelo autómato ou pelo fator humano.

Selecionar, ou seja, se uma ou mais propostas estiverem empatadas, o desempate será escolhido pelo humano ou através de um algoritmo de pontuação pela máquina.

Aprovar, significa que quando apenas uma proposta candidata está sob consideração poderá ser necessário aprová-la manualmente.

Os parâmetros descritos descrevem uma elevada taxa de eficiência na procura pela eliminação de resultados falsos-positivos e falsos-negativos, que se revela uma mais valia para o grau de confiabilidade de qualquer processo que os implemente. No entanto, com uma maior interação humana, o fator tempo desde o início de qualquer processo até ao seu final vai inevitavelmente aumentar.

2.7 Automatismo Total versus Automatismo Parcial

Para a avaliação destes dois tipos de automatismo, apresentamos a tabela 2.5, que contém as vantagens e desvantagens de ambos os processos.

Automatização Total	**Automatização Total**
Diminuição da mão de obra humana	Maior precisão na exploração de vulnerabilidades
Maior Eficiência	Maior controlo sobre as ações
Maior tempo de atividade	

Tabela 2.5: Automatismo Total versus Automatismo Parcial.

Podemos constatar que para um ambiente no qual seja necessária uma maior credibilidade dos resultados finais obtidos, a automação a prevalecer será a Automatização Parcial. Por outro lado, num ambiente em que seja necessária uma maior taxa de eficência e maior rapidez no processo, a Automatização Total é a escolha ideal.

Ao utilizar a Automatização Parcial podemos então tirar partido das vantagens que residem na vertente humana dos ataques descritos anteriormente e na sua correta identificação e execução, utilizando um processo de automatização no que toca à replicação de ataques utilizados com características semelhantes no futuro.

Capítulo 3

Estado da Arte

Neste capítulo são abordados projetos orientados para providenciar uma solução aos problemas descritos anteriormente, também como são abordadas as possíveis ferramentas e *frameworks* necessárias para desenvolver uma solução confiável e eficaz.

3.1 Frameworks existentes

Na presente secção irão ser avaliadas as ferramentas presentes atualmente que visam providenciar uma solução integral ao problema descrito, assim como as suas vantagens e desvantagens.

3.1.1 *metasploit-autopwn*

O *metasploit-autopwn* é um plugin para a ferramenta *metasploit* [25]. Este plugin auxiliar tem como principal objetivo a exploração automática de vulnerabilidades detetadas pela framework *metasploit* através de um sistema de expressões regulares presentes nos serviços detetados.

Esta deteção pode ser manipulada pelo alvo, uma vez que a mesma se baseia num conceito de *banner-grabbing* [53], que é uma técnica comum na área da ciber-segurança que se baseia na deteção de informações adicionais sobre os serviços ativos existentes num determinado alvo pela captura do cabeçalho no protocolo de comunicação. Ao alterar o cabeçalho de comunicação, é possível despistar completamente o *plugin* para a execução de ataques para outra arquitetura ou até mesmo serviço, neutralizando o seu efeito por completo.

Todo o funcionamento consiste em atacar um único alvo (um único IP), com todos os vetores de ataque na base de conhecimentos do "metasploit" para um determinado serviço [24], o que irá originar uma baixa eficiência e irá gerar tráfego desnecessário devido à sua identificação ser demasiado ampla e forçando o teste a vetores de ataque que claramente não se adequam ao alvo em questão.

Salientamos que o *metasplit-autopwn* foi desenvolvido com uma meta final diferente da

utilizada neste projeto. No entanto, a utilização do mesmo encontra-se restrita pelo extrema ineficiência e pelas limitações de alvos possíveis de testes em simultâneo.

3.1.2 Nessus

A ferramenta *Nessus* é um "vulnerability scanner", que como a descrição nos indica, analisa vulnerabilidades presentes nos sistemas alvo ao qual se dirige, através de um extenso escrutínio sobre os serviços e as suas respetivas versões, num determinado alvo [48]. As principais funções do *Nessus* são:

- Identificar vulnerabilidades que conduzam ao acesso indevido;
- Falhas de configurações, sejam estas de ficheiros ou falta de atualizações;
- Passwords fracas;
- Possíveis ataques de Denial of Service (DoS) ou Distributed Denial of Service (DDoS).

Todos estes pontos que a ferramenta pretende identificar possuem uma vertente de alarme indispensável à segurança de qualquer sistema informático. Contudo, após a sua deteção não há indicadores de que existam mesmo vulnerabilidades, uma vez que nunca é efetuada uma exploração do potencial vetor de ataque detetado [26]. Apenas é identificada uma possível vulnerabilidade ou falha de segurança, através do número de versão presente no software existente. Esta verificação a *passwords* fracas, é efetuada através de uma leitura sobre a política de passwords em questão, analisando as regras presentes no ambiente, ou seja, apenas irá verificar políticas, nunca tentando efetuar um processo de login.

O facto desta ferramenta não possuir um processo de exploração de vulnerabilidades, no seu estado de "pós-análise", inutiliza a mesma para o âmbito pretendido, visto que poderão existir potenciais casos de falsos-positivos e falsos-negativos devido à falta de execução dos ataques ao alvo em questão.

3.1.3 arpag

Arpag pode ser entendido como uma referência turca à palavra "magia" ou à palavra "magicamente" [51]. O seu objetivo principal é a automatização da "exploração de vulnerabilidades" e foi desenvolvida por Yelken [62].

O seu método de funcionamento é semelhante a todas as ferramentas do ramo: remete-se à indicação de um IP e porto alvo para os mesmos serem identificados e posteriormente testar possíveis vetores de ataque consoante o resultado fornecido pelo alvo.

Esta ferramenta, embora se comprove muito útil para o arsenal de qualquer analista de segurança, foi descontinuada na sua página ([62]), o que torna impossível a sua distribuição,

mantendo-se assim apenas disponível para quem possua previamente uma cópia da mesma.

Como é possível observar 3.1, existem várias desvantagens, que são provenientes da falta de desenvolvimento contínuo, o que torna a ferramenta obsoleta pelos standards de hoje em dia. Visto que algumas das desvantagens desta tecnologia encontram-se em conflito com os objetivos propostos para este projeto, a ferramenta "arpag" não é viável para este caso de uso.

Vantagens	**Desvantagens**
Automatização de ataques	Software descontinuado e sem manutenção
Utilização da framework "metasploit"	Ataques atribuídos a um alvo específico
-	Adição de ataques requer alterar o código fonte
-	Código pouco eficiente

Tabela 3.1: Vantagens e desvantagens da framework Arpag.

3.1.4 AutoSploit

O *AutoSploit* é uma ferramenta de segurança que apresenta uma elevada complexidade e versatilidade [45]. É uma referência na área da automatização de ataques informáticos, devido à sua obtenção de alvos a partir de listas de acesso público como shodan.io, zoomeye, censys, entre outros, agilizando assim a procura de possíveis alvos para o atacante.

Esta ferramenta possui várias formas de funcionamento, sendo que a sua principal função é a exploração de vulnerabilidades em massa com alvos previamente adquiridos através de *scanners*, ou seja, através da utilização do *AutoSploit* o utilizador da ferramenta deverá indicar também o vetor de ataque a utilizar para os diversos alvos encontrados.

Contudo, a utilização desta ferramenta tem o funcionamento reverso do pretendido para este projeto. A ferramenta *AutoSploit* irá procurar alvos que sejam detentores da tecnologia/software a explorar, ou seja, não irá testar uma lista de alvos específica, mas sim irá efetuar a sua procura pela internet com o auxílio de scanners. Para além da restrição previamente indicada, existe também o facto de não conseguir efetuar a exploração de vários vetores de ataque em simultâneo, limitando-se a um ataque único por cada "lista de alvos".

Podemos então assumir que para além de não cumprir os requisitos propostos, é também uma ferramenta orientada para outro tipo de atividade que por sua vez é o reverso da prevenção de ataques informáticos.

3.1.5 InfectionMonkey

A ferramenta *InfectionMonkey* durante o seu funcionamento, é composta por dois componentes principais, sendo os mesmos denominados por *Monkey Island* e *Monkey* [23]. O *Monkey Island* é um *software* que necessita de ser integrado dentro de uma rede interna previamente existente, que irá então tornar-se o alvo dos ataques desta plataforma. Por outro lado, o *Monkey* detém a função de análise, controlo e demonstração de resultados recolhidos por uma ou mais instâncias

de *Monkey Islands*.

O método de funcionamento desta infraestrutura consiste na integração de uma *Monkey Island* numa rede ou *data-center* onde esta instância irá atacar todos os nós que se encontrem a qualquer nível hierárquico (acima do nível atual da *Monkey Island* ou inferior) e também irá ficar encarregue de executar ataques de movimento lateral. A *Monkey Island* efetua o máximo de movimentações ao seu dispor dentro de uma determinada rede, com a utilização de vetores de ataque comuns e devidamente documentados, que têm de ser "escolhidos" e atribuídos à mesma através da plataforma *Monkey*, ou seja, todas as ações das diversas *Islands* são premeditadas e controladas.

Após terem sido executados todos os testes, irá ser elaborado um relatório para a instância de monitorização *Monkey*, onde o mesmo poderá ser analisado para uma rápida identificação de todas as falhas presentes na rede em questão.

Em suma, o *InfectionMonkey* é uma ferramenta de referência na área da automatização de ataques à segurança informática. No entanto, não corresponde aos critérios necessários para solucionar o problema proposto inicialmente, visto que os seus ataques limitam-se a uma rede interna e exigem uma obrigação para o utilizador de deter um componente adicional e potencialmente malicioso na sua própria rede interna.

3.2 Avaliação de *frameworks* e sumário da análise

As *frameworks* existentes foram elaboradas para outros fins e com o objetivo de atingir resultados diferentes dos pretendidos neste projeto. Contudo, surge a necessidade de uma elaboração de um conceito novo para satisfazer todos os requisitos solicitados: uma ferramenta que apresente resultados consistentes, que seja capaz de atacar vários alvos e tecnologias distintas, e efetuar a sua devida classificação. Uma vez que a junção de funcionalidades de todas estas *frameworks* não é uma opção viável nem estável.

Foi efetuado um aproveitamento de técnicas e metodologias de ferramentas previamente existentes, uma vez que a área da exploração automática de vulnerabilidades já se encontra num certo patamar de maturidade e com conceitos chave a serem utilizados.

3.3 Ferramentas existentes

Na atualidade existem diversas ferramentas como linguagens de programação ou bases de dados. São normalmente utilizadas como base para a programação de qualquer infraestrutura e/ou software. Na presente secção serão abordadas as diversas opções que estão disponíveis e devidamente avaliadas, face aos requisitos solicitados.

3.3.1 Linguagens de Programação

As linguagens de programação descritas abaixo necessitam de estar em conformidade com interoperacionalidade exigida atualmente no desenvolvimento de infraestruturas. Devem também ter uma fácil portabilidade entre sistemas, visto que poderão sofrer uma migração em qualquer instante, na necessidade de mais recursos com o aumento de dimensão da superfície de ataques [58].

3.3.1.1 C

O *C* é uma linguagem extremamente versátil, utilizada para as mais diversas finalidades, desde desempenhar funções vitais em sistemas operativos a aplicações web [47]. Esta versatilidade deriva do facto da mesma ter como objetivo a portabilidade de todos os programas compilados (ou proceder novamente à sua compilação com alterações mínimas ao *source code*).

A linguagem de programação *C* é atualmente uma das referência no que toca a eficiência, tanto a nível de processamento como preservação e alocação de memória [30].

No entanto, é uma linguagem que necessita de um compilador para transformar o *source code* num *object file* para mais tarde ser utilizado um *linker* que irá proceder à ligação entre o posteriormente referido *source code* e *object file*, criando assim finalmente o ficheiro executável que irá correr o programa pretendido.

3.3.1.2 Ruby

Ruby é uma linguagem de programação de alto nível, que reside na categoria linguagens de "interpretação", pelo que não precisa de ser compilada para a execução do *source code* [18]. Contém dentro das suas diversas funcionalidades o mapeamento automático de memória e o facto de ser uma linguagem orientada a objetos, facilitando assim a utilização de vários tipos de estruturas de dados. Esta linguagem foi desenvolvida com o objetivo de ser utilizada prioritariamente para a criação de *scripts*, sendo por sua vez comparada a linguagens como Python e Perl.

Uma análise rápida a algumas *Frameworks* de segurança revelam-nos que o seu *source code* é escrito maioritariamente em *Ruby*, como por exemplo, o metasploit. A utilização desta linguagem facilita a integração com software previamente existente, visto que irão partilhar a mesma linguagem e assim irão simplificar as comunicações entre processos.

Outro ponto positivo em relação a esta linguagem de programação é o facto de a mesma também possuir várias bibliotecas que facilitam o desenvolvimento na área da ciber-segurança, visto ser uma linguagem utilizada na comunidade. Este fator permite uma maior versatilidade na utilização de código orientado à *exploitation* de vulnerabilidade.

3.3.1.3 Python

Python é uma linguagem interpretada e de alto nível, não necessitando de processos de compilação, podendo assim ser executada diretamente de um *script*. Dentro das suas várias funcionalidades, temos também que *Python* é uma linguagem orientada a objetos [50].

Uma grande vantagem também adjacente ao uso desta ferramenta é a sua portabilidade, visto que qualquer software desenvolvido permite a execução em qualquer plataforma através do uso do mesmo script, sem qualquer alteração.

Existe uma vasta quantidade de bibliotecas disponíveis que facilitam a implementação dos vetores de ataques mais comuns, devido a esta linguagem ser atualmente a mais utilizada entre os intervenientes do campo da ciber-segurança.

Esta ferramenta tem também ao seu dispor a utilização de linguagens de um nível mais baixo que o seu, como *C* e *C++* [3], para a eventualidade de ser necessária uma conexão com diferentes softwares quando o mesmo for necessário.

3.3.1.4 Comparação de frameworks

Dadas as tecnologias previamente descritas, apenas uma será utilizada para alcançar o âmbito pretendido, logo terá de ser feita uma avaliação na qual o objetivo será o cumprimento dos requisitos descritos na secção anterior3.3.

Tal como verificamos previamente, *C* é uma linguagem otimizada em termos de processamento e memória. No entanto, a sua falta de bibliotecas aumenta exponencialmente a taxa de esforço necessária para a implementação de vários vetores de ataque. Temos também que as outras linguagens de programação apresentam uma maior portabilidade devido à sua facilidade de execução derivado do facto de as mesmas não necessitarem de ser compiladas e poderem ser executadas diretamente através do *source code*. Todos estes aspetos tornam-se um obstáculo impeditivo à utilização do *C* como linguagem predominante. A funcionalidade que distingue a linguagem *C* de todas as linguagens de alto nível é o facto de a mesma possibilitar a utilização de instruções de baixo-nível.

A má gestão automática de memória desempenhada pelo *Ruby* faz com o que o mesmo tenha vários *overheads* na memória, tornando fúteis todas as tentativas de otimização nesta linguagem, tal como foi comprovado por Abeyrathne [1]. *Ruby* tem também uma carência de bibliotecas quando comparado com *Python*.

Por último, vimos que *Python* é atualmente a linguagem com a maior utilização em termos de ciber-segurança, derivado à sua vasta adoção dos intervenientes desta área, o que contribui para a criação de novas bibliotecas e utilidades da mesma.

Temos também o facto de a mesma ter sido concebida para favorecer a compreensão rápida do utilizador/programador sobre o código descrito de qualquer software nele contido, abdicando

da velocidade e expressividade. Derivado a este factor, o código em *Python* é substancialmente mais curto e conciso quando comparado com outra linguagem, para a mesma funcionalidade.

Concluindo a análise previamente iniciada, a linguagem predominante na elaboração da solução, será *Python*, uma vez que a prévia implementação de bibliotecas para as funções a desempenhar retira vários problemas impeditivos no que toca à solução esperada. A fácil integração com a maior parte dos *scripts* já existentes, distribuídos por fontes públicas para a sua posterior alteração e execução também é um ponto fulcral para esta decisão.

3.3.2 Bases de Dados

Atualmente existem dois tipos de bases de dados: relacional e não-relacional. Uma base de dados relacional está organizada em tabelas, para poder servir de suporte para as relações estruturais sobre os dados que armazena, ou seja, as tabelas podem estar dependentes e/ou relacionadas entre si. Por outro lado, uma base de dados não-relacional está restrita a um único documento, não fornecendo relações entre os dados nele contidos, apenas armazenando informação. De seguida, apresentamos um exemplo de base de dados para cada um destes tipos.

3.3.2.1 MySQL

MySQL é um sistema de gestão de bases de dados, que utiliza a linguagem Structured Query Language (SQL) como uma interface para a modulação e persistência de dados [40]. Uma vez que se trata de uma base de dados relacional, os dados estão hierarquicamente ligados entre si, sendo possível cruzar referência com diversas tabelas e efetuar a agregação e correlação de resultados. O fator de relação e hierarquia torna este tipo de base de dados especialmente útil na agregação de conceitos diretamente relacionados, visto que poderão ser obtidos através de uma *query* SQL.

3.3.2.2 MongoDB

O *MongoDB* [28] é um software de base de dados não-relacional, orientado a documentos. Cada documento presente neste tipo de base de dados corresponde a todas as instâncias de dados guardadas por um programa e/ou infraestrutura.

A utilização deste gestor de base de dados, permite o *nesting* da informação, que consiste em agrupar a informação seguindo uma hierarquia previamente definida dentro de um *schema*. O retorno destes dados é normalmente efetuado no formato JSON, o que facilita a sua portabilidade e comunicação inter-plataformas, sendo este mesmo formato abstrato de qualquer linguagem de programação.

3.3.2.3 Comparação de bases de dados

No desenvolvimento da solução pretendida, será necessário o armazenamento de vários dados relativamente aos alvos, clientes e *scripts* a utilizar.

Devido à natureza entre as relações de "alvos" e clientes, teremos uma relação 1:N visto que teremos como um cenário previsto um cliente possuir vários alvos. Este factor promove a escolha de uma base de dados relacional como a solução ideal para a proposta apresentada, uma vez que a mesma pode portar todos os conceitos hierárquicos necessários para o posterior desenvolvimento [44].

Quanto ao armazenamento de *scripts*, serão armazenados no sistema de ficheiros do sistema, sendo carregados para memória com o objetivo de posteriormente ser efetuado um mapeamento entre todos os seus componentes e descrições. A utilização do formato JSON será utilizada com o objetivo de facilitar a partilha de dados com componentes futuras, e para criar uma camada de abstração no carregamento e criação de *scripts* de ataque.

3.4 Utilização da solução no mundo real

No decorrer da delimitação da ideia de criar uma plataforma para mitigar os riscos e vetores de ataque conhecidos atualmente, foi efetuada uma pesquisa, com a qual se pode concluir que a divisão dos ataques informáticos referentes a fugas de informação poderá ser dividida em cinco grandes vetores [59]:

- 45% envolveram *hacking*;
- 22% envolveram erros casuais e pontuais;
- 22% envolveram ataques sociais;
- 17% envolveram *malware*;
- 8% envolveram utilização não cuidada de utilizadores.

Os dados acima referidos, revelam uma forte tendência para a automatização no que toca a ataques à segurança informática de várias empresas. Estas técnicas de automatização e exploração de vulnerabilidades são utilizadas atualmente com base na aplicação de *scripts* para tomar partido dos seus alvos o mais rapidamente possível, antes que os mesmos consigam ser corrigidos ou atualizados pelo respetivo administrador de sistemas. Isto traduz-se numa autêntica "corrida contra o tempo", entre atacantes distribuídos e/ou organizados pela internet e pelos detentores da informação.

Salientamos também, que os números atuais de ciber-ataques estão a ultrapassar limites históricos, com aumentos aproximadamente de 300%, desde que a pandemia relativa ao Covid-19

foi decretada a nível mundial [59]. Desde então, começaram a surgir atores maliciosos de todas as partes do mundo, numa tentativa de capitalizar sobre as empresas a utilizar o regime de trabalho remoto. Para além deste fator, com a quarentena obrigatória, foram reunidas várias condições que facilitam a exploração de vulnerabilidades em massa.

Esta "corrida" à descoberta de vulnerabilidades e falhas de segurança informática na qual os participantes maliciosos não possuem qualquer moralidade ou restrição quanto aos ataques efetuados, origina uma realidade temporal no que toca à descoberta das ditas vulnerabilidades por parte da equipa de segurança que pretende manter os sistemas protegidos e os respetivos atacantes.

Assim sendo, é criada a necessidade de uma ferramenta de análise eficiente e precisa que irá incidir sobre todos os *assets* de uma determinada empresa, da mesma maneira que um atacante comum/automatizado, numa tentativa de identificar e corrigir vulnerabilidades antes que as mesmas sejam detetadas e exploradas por terceiros com os objetivos destrutivos, referidos anteriormente.

É de ressalvar que análises periódicas não resultam numa mitigação total das vulnerabilidades que possam estar expostas, embora seja uma ferramenta essencial para ser criada uma "vantagem" no que toca à defesa de ativos, visto que uma emulação de um atacante real é um dos pontos principais deste projeto.

Para efeitos de análise e pesquisa sobre o cenário real de ataques, temos por exemplo a ferramenta elaborada por Robert David Graham [22] que efetua um *scan* por um dado intervalo de endereços de IP e verifica se os alvos procurados são vulneráveis ao "CVE-2019-0708" mais conhecido por *BlueKeep*. Este scanner pode ser utilizado em junção com outra ferramenta para acelerar o seu processo [21]. No entanto, no âmbito de provar todos os pontos expostos anteriormente, foi efetuada uma pesquisa no dia 19 de julho de 2021 na qual os objetivos foram os seguintes:

- Identificar máquinas com software potencialmente desatualizado;
- Menor tempo possível de pesquisa;
- Elevada taxa de fiabilidade;
- Identificar máquinas nas quais o "CVE-2019-0708" possa ser utilizado.

Através da utilização de uma plataforma elaborada para este efeito (shodan.io [32]), agregada com os benefícios de uma conta portadora de *Membership*, foi possível identificar em cerca de 3 segundos 72 máquinas vulneráveis que se encontram dentro dos critérios pretendidos, como podemos consultar na imagem 3.1 (a imagem foi censurada propositadamente para evitar a divulgação de dados potencialmente sensíveis):

Toda a informação e ferramentas utilizadas, embora algumas com custos mensais/anuais (exemplo: Shodan.io) apresentam uma grande facilidade quanto à automatização de um vetor de

Figura 3.1: Análise utilizando a plataforma Shodan para identificar alvos potencialmente vulneráveis.

ataque específico e à sua utilização. Os dados podem ser recolhidos através destas plataformas para serem adicionados a uma extensa e complexa lista de possíveis alvos por parte dos agentes maliciosos, que irão por sua vez percorrer uma grande gama de alvos distribuídos por toda a internet, em busca de máquinas para diversos fins que lhes sejam proveitosos, ignorando qualquer barreira até alcançarem o seu objetivo.

Podemos afirmar que pelo extenso arsenal que os atacantes possuem ao seu dispor, todos os elementos que trabalham na defesa estão atualmente num estado de alerta constante face aos novos vetores de ataque que se encontram a surgir, visto que entre a data de publicação e a respetiva data de resolução do problema (mesmo que sejam apenas alguns dias), já terão sido feitos enormes estragos em infraestruturas empresariais e/ou estatais, visto estas serem os alvos mais procurados entre os agentes maliciosos.

A utilização de uma validação rápida e eficaz de alvos prioritários, nomeadamente "clientes" ou "utilizadores", resulta numa diminuição no tempo de reação por parte da equipa de segurança responsável pelos ativos a proteger. No entanto, para isto se revelar possível, é necessária a elaboração de um *script* com o novo vetor de ataque encontrado, em função das métricas exigidas pelo projeto desenvolvido, para o mesmo rapidamente determinar se os ativos se encontram vulneráveis, acelerando assim a contra-resposta da equipa à possível exploração da máquina-alvo, ou até mesmo em casos extremos efetuar um *patch* temporário para a sua mitigação.

Capítulo 4

Desenho e Desenvolvimento

No presente capítulo será efetuada uma análise e exposição da proposta de solução incluindo o desenho da infraestrutura e implementação bem como o seu funcionamento.

4.1 Análise de Requisitos

A infraestrutura exposta como solução apresenta vários conceitos e funcionalidades distintas em todo a sua execução, derivadas da modularidade dos componentes. Modularidade essa adquirida pela separação dos conceitos apresentados, sendo possível o reaproveitamento dos *scripts* para testes isolados, visto que os mesmos se encontram segregados num diretório diferente e podem ser invocados sem recorrer a toda a infraestrutura. Este aspeto apresenta várias vantagens no que toca à verificação manual (quando necessária), uma vez que disponibiliza as ferramentas utilizadas de uma forma direta e concisa.

Outra das funcionalidades chave é a escalabilidade, pelo que se torna possível através da utilização de subprocessos para efetuar as funções de ataque, acompanhando assim o aumento no número de clientes e tarefas a efetuar paralelamente.

A unificação ou padronização de parâmetros facilita também futuras adições na base de dados de *scripts*, uma vez que todos os modelos de ataque devem seguir uma lógica própria para o correto funcionamento da arquitetura proposta, tendo as respetivas *exit messages* e outro controlo de erros sendo inteira responsabilidade dos *scripts* e não da infraestrutura.

A automatização de ataques é o objetivo final de todo o projeto, uma vez que irá facilitar a análise a uma grande superfície de alvos (com o seu devido consentimento e aprovação) enquanto comprova todas as possíveis falhas de segurança, utilizando os vetores de ataque previamente persistidos na infraestrutura.

É efetuado um registo de todas as tentativas de ataque, sejam tentativas bem-sucedidas ou não, com a finalidade de providenciar um registo sobre as vulnerabilidades encontradas para uma futura revisão e análise. É utilizado este registo também num âmbito de salvaguardar o

utilizador do sistema, pois todas as suas ações serão persistidas.

Num contexto de avaliação, temos a funcionalidade de pontuação, na qual são atribuídos pontos a um cliente com base no seu desempenho face aos testes de segurança efetuados. É de salientar que a pontuação tem um grau de criticidade crescente, sendo 1 o menos grave e 10 o nível mais crítico.

4.2 Design da Infraestrutura

A infraestrutura para o correto funcionamento modular da proposta de solução necessita de uma base de dados relacional para a persistência dos dados e as suas respetivas relações hierárquicas. Além disso, é necessária a coordenação dos ataques informáticos que irão ser lançados, o que suscita uma necessidade de existir um caminho de rede (interna ou internet) válido até ao alvo.

Toda a comunicação efetuada entre o Sistema de Gestão de base de dados relacional é efetuada através de uma biblioteca de Python. A comunicação entre a infraestrutura de ataque e o alvo por outro lado, oferece uma maior diversidade no que toca a protocolos de comunicação, podendo o mesmo efetuar vários tipos de pedidos (Hypertext Transfer Protocol (HTTP), sockets, ente outros). Não podem ser atribuídos protocolos de comunicação específicos nesta parte da infraestrutura devido à sua volatilidade e adaptabilidade ao sistema alvo e a sua respetiva vulnerabilidade.

Uma vez que foi adotado um conceito de "armazenamento local de *scripts*", no qual a relação entre o *script* em questão e as restantes informações sobre os ataques a realizar encontram-se guardadas num ficheiro que deverá ser carregado na execução do programa.

As comunicações referidas anteriormente seguem o diagrama da figura 4.1.

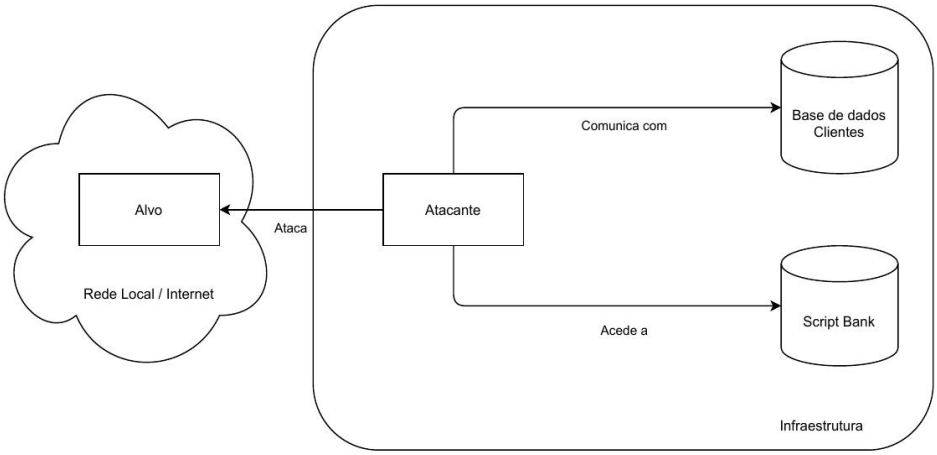


Figura 4.1: Diagrama funcional da infraestrutura.

4.3 Fluxo de informação

Nos seguintes subcapítulos serão expostas as recolhas de dados e as inserções das mesmas nas respetivas bases de dados relacionais e ficheiros, para o correto funcionamento do programa no ambiente de testes.

4.3.1 Base de dados relacional

A estruturação inicial da base de dados, num primeiro desenvolvimento de encontro aos requisitos propostos inicialmente, consistia em apenas numa tabela, com as respetivas informações dos clientes e os seus respetivos alvos. Esta solução provou ser ineficiente dado que poderão existir futuros clientes que utilizem o mesmo recurso, tornando assim necessário repetir o registo do alvo para vários clientes. Esta ideia foi então substituida por uma relação hierárquica entre os clientes e os alvos para facilitar a reutilização de informação e evitar duplicações.

Após a identificação deste problema, a base de dados sofreu uma alteração estrutural, criando assim a sua segunda iteração, que consistia em duas tabelas, uma para a identificação de alvos e outra para a identificação de clientes. As duas tabelas são relacionadas através de uma relação de chave estrangeira, associando uma relação de 1 cliente a N *alvos* (1:N).

De seguida, num âmbito de preservar os interesses dos clientes e providenciar registos para controlo interno, foi necessário elaborar um sistema de pontuação e de *logging* sobre os ataques efetuados. Estes requisitos originaram novamente numa alteração da estrutura da base de dados, acrescentando duas novas tabelas: *log_attack* e *scoring*. A função destas novas tabelas é registar todos os ataques efetuados e guardar a pontuação obtida pelos clientes ao longo do tempo, respetivamente. Podemos verificar os atributos e as relações partilhadas entre as tabelas pela imagem 4.2.

Na tabela *clients*, temos uma referência ao nome do respetivo cliente e o seu "id" para posteriormente ser possível a sua identifcação no decorrer do cruzamento de informação com as restantes tabelas.

A tabela *scoring* possui uma ligação direta ao cliente, visto que a responsabilidade desta tabela é a persistência dos dados referentes à avaliação do cliente, marcando para uma determinada data (contida no campo "ts_score") a pontuação mais elevada de uma determinada vulnerabilidade e a respetiva média (presente na coluna "average").

Quanto aos dados da tabela *targets*, cada registo é composto por um endereço ou caminho que permita a correta identificação e mapeamento até á máquina alvo, e por sua vez, a respetiva *port*, visto que estes dados são necessários para a identificação do serviço a explorar. Além disso, temos também a identificação do serviço alvo, para mais tarde ser avaliado contra o armazenamento local de *scripts*. Esta tabela relaciona-se diretamente com a tabela *clients* através de uma relação de chave estrangeira.

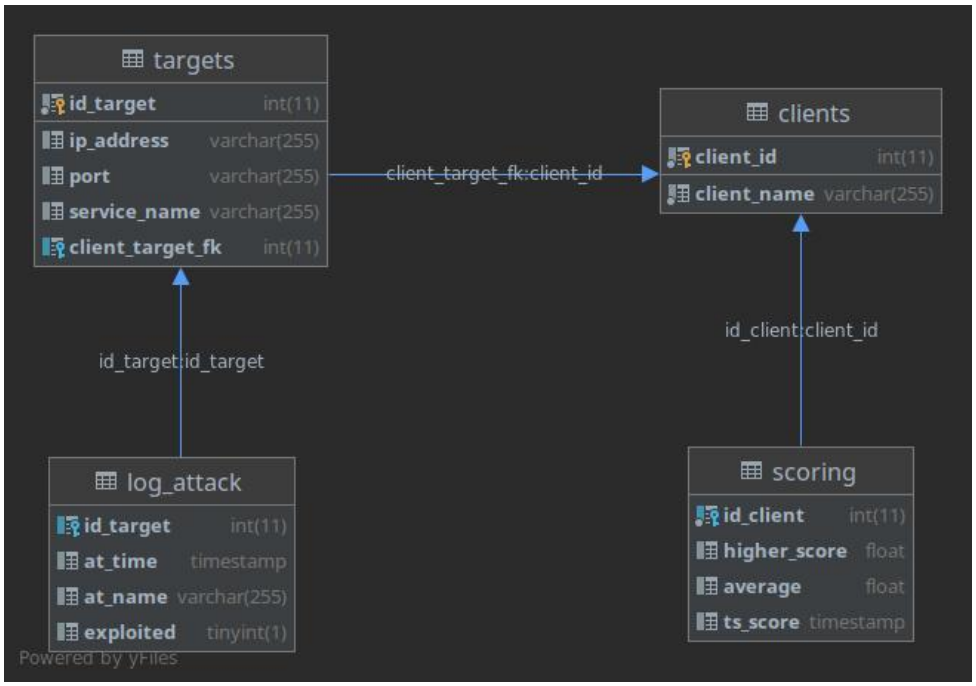


Figura 4.2: Diagrama da base de dados.

Por fim, temos a tabela *log_attack* cuja função é persistir os dados do ataque efetuados, marcando a hora da execução do ataque (através do campo "at_time") do respetivo *script* utilizado para efetuar o ataque ("at_name") e o campo *"exploited"* indica-nos através da atribuição do valor 0 no caso de o alvo ser atacado sem sucesso ou 1 caso contrário.

4.3.2 Banco de *Scripts*

O conceito de banco de *scripts* foi inicialmente adotado para favorecer os benefícios previamente apontados sobre o automatismo parcial, uma vez que os mesmos sendo desenvolvidos por uma componente humana, irão providenciar uma maior taxa de sucesso na deteção de vulnerabilidades e por sua vez reduzir todos os "falsos-negativos" e "falsos-positivos" que advêm de toda a análise.

Considerando os diversos vetores de ataque presentes atualmente no mundo da ciber-segurança, é necessário garantir uma correta execução para todo o tipo de tecnologias, o que obriga a uma abstração do banco de *scripts*, de modo a que este se consiga adaptar a qualquer caso ou alvo, enquanto simultaneamente permite desenvolvimentos futuros e integrações com outras tecnologias/*frameworks*.

Na figura 4.3 podemos verificar a estrutura utilizada, sendo que a mesma se remete sobre os seguintes objetos:

- name_script

  Breve descrição da vulnerabilidade a testar, para posterior identificação na base de dados

Figura 4.3: Estrutura banco de scripts.

quando a mesma tiver de sofrer a persistência do seu registo para facilitar a visualização de informação.

- path

  Este atributo remete-nos para o path no sistema operativo no qual se encontra o *script* a executar. Este campo terá de conter um caminho válido no qual as instruções a executar sigam os padrões utilizados pela infraestrutura.

- exploit_info

  No campo exploit_info deveremos encontrar uma identificação das versões válidas ou uma "tag" que indique que o sistema alvo está vulnerável. Este campo é decisivo na execução ou não do ataque, uma vez que o mesmo verifica este registo com os dados selecionados anteriormente da base de dados relacional. Se estes partilharem a mesma "tag", então pressupõe-se que o alvo está vulnerável e o mesmo terá de ser sujeito ao *script* de ataque correspondente.

- cvss

  O valor no campo de cvss remete-nos para a severidade da vulnerabilidade contida no ataque a desempenhar. Este valor será utilizado posteriormente na classificação dos clientes.

4.4 Implementação

Na seguinte subsecção serão abordados os tópicos inerentes à base de dados relacional utilizada e a sua respetiva estruturação e programação.

4.4.1 Base de Dados

Para o correto funcionamento da base de dados é necessário instalar a componente MariaDB que é um *fork* gratuito do gestor de base de dados relacionais *MySQL* criado pela comunidade [6]. A utilização desta plataforma deriva da sua biblioteca de comunicação com a linguagem de programação *Python* e na utilização dos seus conetores e cursores para a execução de *queries* Structured Query Language (SQL).

Um conetor necessita de ter vários parametros, nomeadamente um utilizador e a sua respetiva senha, que correspondem aos campos *user* e *password* respetivamente. Por outro lado é necessário apontar também o endereço de IP ou domínio no qual a base de dados se encontra integrada com o indicativo da sua respetiva *port* (que por defeito é 3306). Necessita também do nome da base de dados na qual os dados estão persistidos.

Para exemplificar a conexão, a ilustração 4.4 mostra uma chamada à base de dados na tentativa de extrair informação.

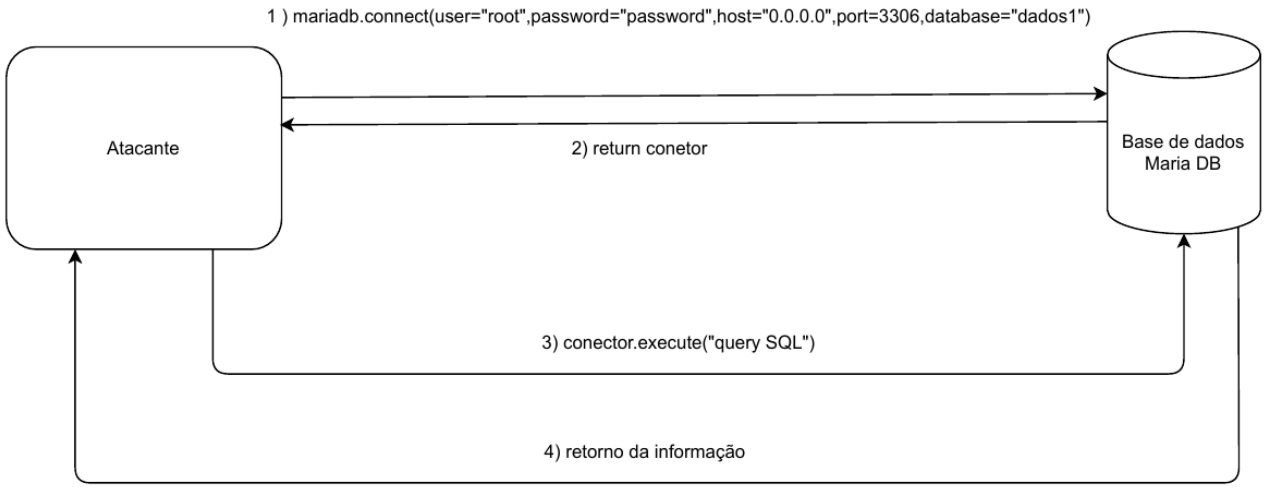


Figura 4.4: Esquema de acesso à base de dados relacional do programa.

É possível verificar no passo número 1) a conexão e autenticação da entidade que requisita acesso à base de dados

No entanto, é também necessária uma inicialização de todas tabelas necessárias e as suas respetivas propriedades relacionais para que o programa consiga ser executado com sucesso, ou seja, as tabelas *target* e *clients* necessitam de uma introdução de dados válidos para serem efetuados os ataques por parte da infraestrutura. Por outro lado, as tabelas *log_attack* e *scoring* devem permanecer vazias, visto que serão preenchidas através do correto funcionamento do programa.

4.4.2 Banco de *Scripts*

A consistência do banco de scripts tem de ser mantida através de um standard e padronização que consiste na delegação da responsabilidade de tratar erros e validar o sucesso ou falha do ataque para o *script* e não para a infraestrutura. Toda a responsabilidade em validações, vetores de ataque, mapeamento de memória (se necessário) ou de páginas *web* deverá ser inteiramente responsabilidade da execução externa.

Ao delegar a responsabilidade no tratamento dos ataques, podemos atingir uma maior generalização na infraestrutura, permitindo então a execução de uma forma universal entre linguagens de programação, facilitando assim algumas implementações e aumentando a versatilidade na adaptação de ferramentas já existentes para adicionar à lista de ataques possíveis.

Os únicos parâmetros invocados como argumentos para a execução de ferramentas é o endereço de IP em formato de *string* no primeiro argumento e em segundo lugar no vetor de argumentos, temos a *port* respetiva do serviço, sendo esta informação previamente fornecida pela base de dados relacional na tabela *targets*.

Na imagem 4.5 podemos observar as relações entre as duas principais instâncias de persistência. O facto de todo este comportamento decorrer de uma forma abstrata e compatível com a maior parte das tecnologias modernas, é possível re-utilizar estes dados sempre que necessários e até integrá-los com outros tipos de plataformas.

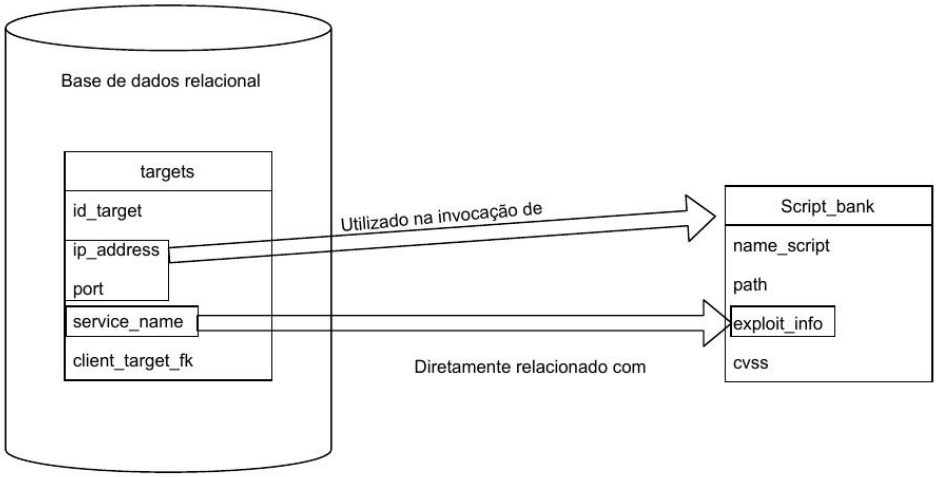


Figura 4.5: Relação entre os conceitos da base de dados e do banco de *scripts*

A utilização do formato de dados JSON presente na junção entre os *scripts* e os seus atributos adiciona uma camada de abstração no que toca a tecnologias e linguagens de programação, permitindo uma integração com um maior número de agentes ou plataformas externas. A

utilização desta formatação de estrutura de dados permite ainda a edição rápida para a adição de novos *scripts*, uma vez que é expectável que seja efetuado uma operação de *append* no ficheiro em questão, de modo a facilitar a adição de futuros vetores de ataque e *scripts*.

4.4.3 Funcionamento da aplicação

O funcionamento da aplicação deverá começar com uma seleção do cliente a analisar, utilizando uma *query*, em que a condição de procura é o seu nome, utilizando os registos presentes na base de dados relacional. Em seguida, será efetuado um mapeamento entre as tabelas *target* e *client* para serem obtidos os alvos específicos do cliente previamente indicado.

A imagem 4.6 remete-nos para a filtragem pelo parâmetro nome do cliente e à devida relação 1:N das tabelas em questão. No seguimento da mesma lógica, é também necessário extrair todas as informações da tabela *target* visto que as mesmas são necessárias para a correta execução do programa.

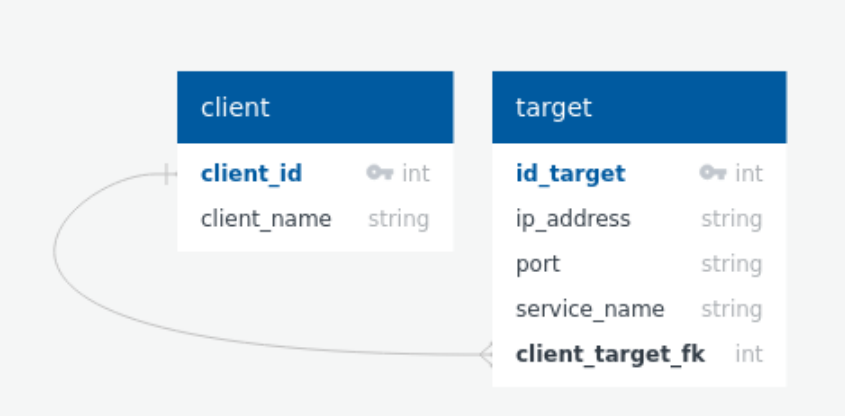


Figura 4.6: Relação entre a tabela client e target na base de dados relacional

Após a extração relacional dos dados, é necessário carregar para memória todo o banco de *scripts* para posteriormente serem efetuados os testes relativos à ciber-segurança dos alvos em questão. Os mesmos serão carregados por um ficheiro denominado ”script_bank.json” que necessita de se encontrar no mesmo diretório que a ferramenta principal. Devido ao seu formato JSON o mesmo será carregado como um ficheiro e importado como variável diretamente sobre o conteúdo lido devido à biblioteca ”json” presente em Python, como se pode ver pelo excerto de código 4.1.

```
import json
...
with open("script_bank.json", "r") as script_bank_file:
    sc_bank = json.load(script_bank_file)
    sc_bank = sc_bank["script_bank"]
```

Bloco de Código 4.1: Carregar o Banco de Scripts.

A utilização desta técnica pressupõe que todas as alterações efetuadas ao ficheiro pelo utilizador se encontram válidas com as especificações base da aplicação, sendo as mesmas carregadas diretamente do ficheiro sem qualquer tipo de verificações adicionais, com o intuito de possibilitar a adição de novas métricas e funcionalidades futuras. Esta funcionalidade poderá ser utilizada com a adição de atributos no ficheiro ”script_bank.json” que mais tarde poderão ser incluídas em novas funcionalidades de acordo com os parâmetros ou métricas adicionais.

Posteriormente, será iniciado um processo iterativo para a comparação dos *exploits* carregados e as respetivas *stacks* tecnológicas utilizadas pelos alvos, numa tentativa de identificar o ataque adequado a cada alvo. Esta comparação tem como objetivo reduzir a carga sobre a rede (interna e do alvo) uma vez que executar todos os ataques conhecidos seria uma má prática e iria reduzir significativamente a performance da infraestrutura. Temos então que após efetuada a adequação dos ataques ao alvo, o mesmo será executado através de uma biblioteca para subprocessos para fazer com que o processo de testes funcionais de vulnerabilidades seja mais eficiente e escalável a longo prazo com a adição de mais alvos e/ou clientes na base de dados relacional e o crescimento de possíveis vetores de ataque a explorar.

Durante a fase de ataques são efetuados *logs* de todas as tentativas, falhadas ou bem sucedidas na exploração de vulnerabilidades, com o respetivo indicativo do tempo do ataque, ataque utilizado e o respetivo alvo atacado, relacionando assim todas as ações efetuadas pela infraestrutura para futura análise e/ou comparação com resultados anteriores para possibilitar a análise na correção das falhas de segurança do cliente.

Toda esta informação é persistida numa tabela que está contida na base de dados relacional e que possui uma relação direta com a tabela de alvos, visto que cada ataque tem obrigatoriamente de ser efetuado a um alvo, partilhando assim uma relação de 1:N entre as tabelas “log_attack” e “target”, pois o mesmo alvo pode ser atacado várias vezes em instâncias temporais diferentes. A relação entre as tabelas está ilustrada na imagem 4.7.

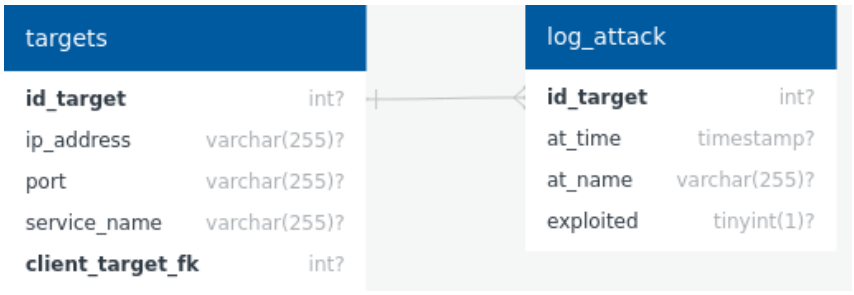


Figura 4.7: Relação entre target e log_attack.

Ainda sob o âmbito da fase de ataques, é necessário o cálculo da pontuação de cada cliente para a sua posterior avaliação, da qual é necessário atribuir consoante a falha ou o sucesso de todo o vetor de ataque um valor. Neste caso concreto, esse valor corresponde à pontuação Common Vulnerability Scoring System (CVSS) que é uma referência no campo da classificação de vulnerabilidade. Este valor está atribuido diretamente ao vetor de ataque utilizado, visto estar contido nos parâmetros de cada ataque inserido no banco de *scripts*. É de realçar que o

valor pretendido para ilustração encontra-se a vermelho, na imagem 4.8.

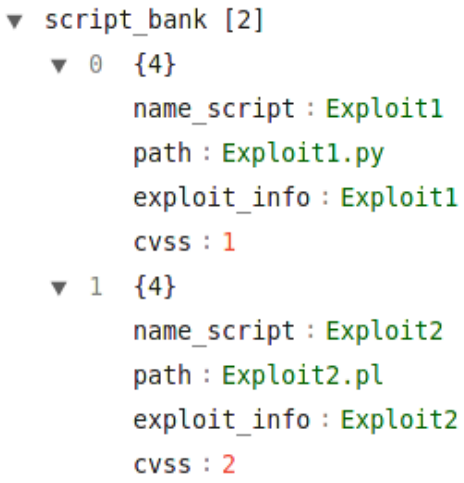


Figura 4.8: Sistema de pontuação
(A pontuação encontra-se realçada a vermelho)

Após o cálculo de uma média ponderada e do registo sobre o maior valor registado, é guardada a respetiva avaliação do cliente na base de dados. Nesta tabela, serão inseridas todas as avaliações sucessivas de todos os clientes, uma vez que assim, será possível efetuar uma análise ao longo do tempo sobre o comportamento e resiliência da empresa alvo a todas as tentativas de quebrar os seus mecanismos de ciber-segurança. De seguida, os dados relativos à pontuação serão persistidos na tabela ”score” que possui a seguinte estrutura e relação com a tabela client 4.9.

Temos que os atributos previamente mencionados (tempo, maior pontuação CVSS, e a média) estão contidos sobre as colunas “ts_score”, “higher_score” e “average” correspondentemente.

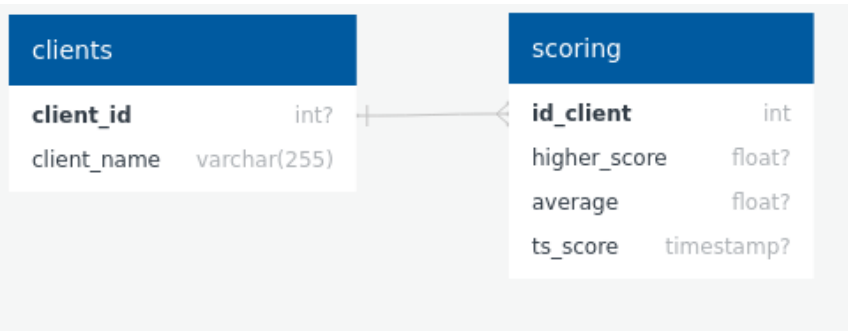


Figura 4.9: Relação entre as tabelas client e score.

4.4.4 Segurança

Dado que toda a infraestrutura se baseia em executar código proveniente de fontes de terceiros e que o mesmo terá de efetuar algumas ações diretamente na máquina alvo e na que executa todos os ataques, a mesma necessita de um elevado grau de confiança no que toca aos possíveis vetores

de ataque carregados pela aplicação. No entanto, realçamos que alguns ataques necessitam também da elevação de privilégios para serem executados com sucesso. Este fator de confiança extende-se também aos alvos, sendo que os mesmos exigem consentimento e que a aplicação irá atacar indiscriminadamente com base nos registo extraídos.

Os pontos apresentados previamente fazem parte de uma base de confiança que não pode ser quebrada, exposta ou de qualquer outra maneira controlada por um ator malicioso, visto que fazem parte dos mecanismos funcionais e principais da aplicação. Estes acessos deverão ser controlados e regidos por uma aplicação de permissões mínimas para o correto funcionamento da aplicação. Um exemplo prático seria a utilização de credenciais seguras para a base de dados de clientes e a alteração de permissões de escrita para o ficheiro "script_bank.json", entre outras.

A segurança da restante infraestrutura em relação a *updates*, manutenções, cópias de segurança, entre outros aspetos, está diretamente relacionada com o utilizador da infraestrutura, sendo a aplicação em si segura, fazendo a responsabilidade desta gestão fora do âmbito aplicacional do sistema.

Capítulo 5

Experiências e Resultados

No presente capítulo é apresentada a composição do laboratório de testes e os dados utilizados para os mesmos bem como os resultados obtidos e respetiva análise.

5.1 Cenário de Testes

Para efetuar os testes para a prova de conceito da solução, foi necessário adicionar dados fictícios às tabelas e criar uma rede para servir de testes à plataforma. Numa tentativa de elaborar sistemas alvo, utilizaram-se várias tecnologias, sendo uma das mesmas o software de virtualização *Docker* [2]. Com o auxílio ao *Docker* é possível emular uma rede e máquinas virtuais para a popular, sendo que esta máquina virtual foi propositadamente deixada vulnerável para o efeitos de teste. Foi necessário também popular a base de dados relacional com empresas fictícias e adicionar *scripts* ao banco para funcionarem como uma validação à correta execução e padronização do sistema.

As vulnerabilidades a explorar foram o "CVE-2014-6271" [41], mais conhecido por "Shellshock" dentro da comunidade de ciber-segurança, que reside numa versão antiga da linha de comandos de sistemas que utilizem o software "bash", a partir da qual um atacante com um pedido específico a utilizar o protocolo Hypertext Transfer Protocol (HTTP) poderá ter acesso com privilégios elevados ao sistema.

O outro *script* utilizado puramente para teste, verifica se o servidor responde corretamente a *pings* ou se os descarta tentando prevenir um vetor específico de Distributed Denial of Service (DDoS) que toma partido da amplificação deste tipo de pedidos.

5.2 Análise de Resultados

Os objetivos alcançados podem ser comprovados com o auxílio a imagens demonstrativas de todos os processos executados. Inicialmente é preciso verificar que processos existem na máquina

alvo, que se verificam pela seguinte imagem 5.1:

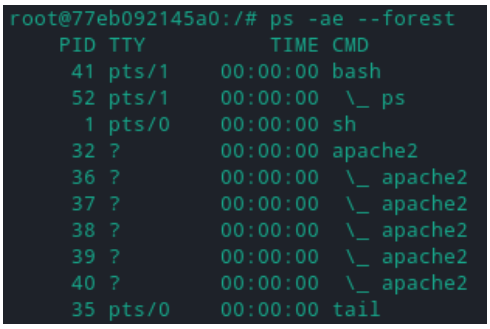


Figura 5.1: Estado do alvo pré-ataque.

Após validar que tudo se encontra como o esperado no alvo, é preciso recorrer à exploração de vulnerabilidades utilizando a infraestrutura desenvolvida. Para efeitos de teste foram utilizados os *scripts* desenvolvidos e indicados anteriormente no capítulo anterior 4.4.3.

Através da exploração da vulnerabilidade "Shellshock" irá ser criada uma conexão com acesso à "Bash" para a execução de comandos arbitrários por parte do atacante que será imediatamente revogada pelo atacante uma vez que o objetivo do mesmo já se encontra cumprido após verificar o retorno da conexão. Ao efetuar uma nova listagem dos processos neste momento, é possível observar a criação de um novo processo (número 54 e 55) como presente na figura 5.2.

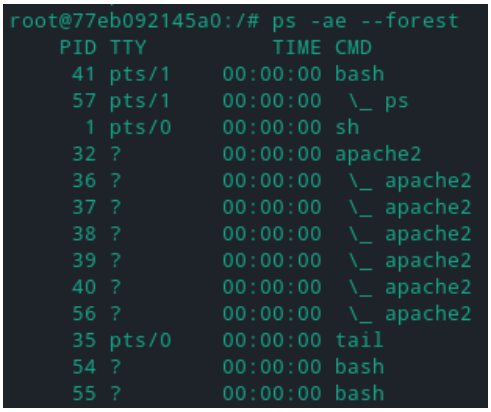


Figura 5.2: Estado do alvo pós-ataque.

De seguida, foi efetuado o teste ao tratamento dos pedidos de *ping* da máquina alvo, como podemos verificar pelo output do software atacante, ambos foram bem sucedidos.

Para a execução destes ataques, foi necessário popular a base de dados relacional com dados referentes ao alvo que se encontra inserido no ambiente de virtualização *Docker*, sendo assim tudo direcionado e testado localmente sobre um ambiente monitorizado e controlado. A população das duas tabelas (5.1 e 5.2) foi efetuada com os seguintes dados:

Após a extração dos dados e a seleção do cliente-alvo com o nome "shell_shock", irão ser utilizados os registos 3 e 4 da tabela 5.2 como vítimas para os testes da plataforma. Estes alvos mencionados previamente portam dados compatíveis com a informação inserida no banco de

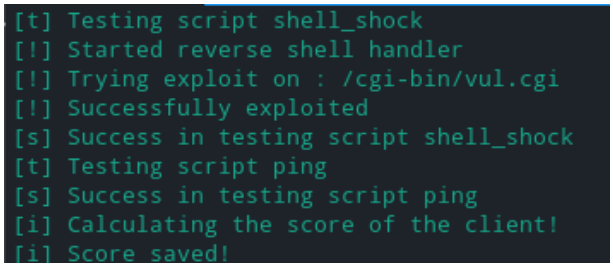


Figura 5.3: Output da Infraestrutura.

client_id	client_name
1	test_company
2	another_company
3	shell_shock

Tabela 5.1: Tabela Clients para testing.

id_target	ip_address	port	service_name	client_target_fk
1	10.9.0.80	80	shell_shock	3
2	10.9.0.80	123	ping	3

Tabela 5.2: Tabela Targets para testing.

scripts para serem corretamente apontados e executados. O "script_bank.json" segue o formato de código presente em 5.1:

```
{ "script_bank": [
  {
    "name_script":"ping",
    "path":"Scripts/ping.py",
    "exploit_info":"ping",
    "cvss": 1.0
  },{
    "name_script":"Exploit_Script_ShellShock",
    "path":"Scripts/exploit_script.py",
    "exploit_info": "shell_shock",
    "cvss": 10.0
  }
  ]
}
```

Bloco de Código 5.1: Banco de scripts para testes.

Os *scripts* utilizados não irão ser fornecidos, uma vez que os mesmos não estão contidos no âmbito deste projeto. Contudo, é possível observar a introdução de parâmetros através do excerto de código seguinte 5.2.

```
...

serversocket = socket(AF_INET, SOCK_STREAM)
```

```
buff = 1024
addr = (lhost, lport)
serversocket.bind(addr)
serversocket.listen(10)
print "[!] Started reverse shell handler"
thread.start_new_thread(exploit,(lhost,lport,rhost,0,payload,pages,))

buff = 1024

try:
   clientsocket, clientaddr = serversocket.accept()
   print "[!] Successfully exploited"

except:
   print "[!] Failed exploit"
}
```

Bloco de Código 5.2: Banco de scripts para testes.

Com uma aceitação do socket para obtermos a conexão remota e o correto encerramento do programa, temos a prova inequívoca que o sistema alvo foi comprometido e verificado como inseguro. Em caso de falha irá ser lançada uma exceção que informa a infraestrutura da falha do ataque.

É de ressalvar que a correta identificação da exploração de vulnerabilidades recai sobre o *script* a ser executado, tendo o mesmo que cumprir a padronização utilizada no decorrer de toda a aplicação, ou seja, não podem ocorrer exceções não previstas durante a execução do ataque e o retorno do *script* final invocado pela infraestrutura terá de retornar o valor 0. Este mecanismo foi adotado, visto que é agnóstico de tecnologias ou linguagens de programação específicas.

Capítulo 6

Conclusões

Neste capítulo serão abordadas as restrições impostas pelas tecnologias atualmente disponíveis que resultam em limitações em toda a infraestrutura desenvolvida no decorrer da realização do trabalho bem como as principais conclusões.

6.1 Limitações no Mundo Real

O mundo real difere em vários aspetos do ambiente de testes utilizada neste relatório, no que diz respeito à utilização de tecnologias distintas das testadas, que restringem os vetores de ataque como:

- Intrusion Prevention System (IPS)

  Previne conexões e a devida filtragem de pacotes potencialmente perigosos para a máquina alvo.

- Firewall

  Previne conexões indevidas no que diz respeito a um determinado ativo que se encontre dentro da rede protegida pela mesma.

- Reverse Proxy

  É um sistema intermédio que recebe os pedidos Hypertext Transfer Protocol (HTTP) direcionados a aplicações web ou APIs e está localizado no ponto final de conexão de qualquer rede. É utilizado para criar uma barreira entre os pedidos de utilizadores e o servidor disponível para a rede.

- Honeypots

  Funcionam essencialmente como uma ilusão (ou isco) para um determinado atacante e são normalmente configurados até ao mínimo detalhe para se assemelharem à infraestrutura e às máquinas adjacentes numa tentativa de despistar o potencial atacante.

Salientamos que as tecnologias apresentadas previamente são apenas ilustrativas das imensas possibilidades existentes e que cada infraestrutura é única em metodologias e sistemas de defesa. Por esta razão, existe uma grande taxa de esforço em orquestrar qualquer ataque sempre da mesma forma ou até mesmo em certos casos semelhantes, podemos ter resultados diferentes só pela disposição da rede, ou pela infraestrutura que se encontra do lado defensor, ou simplesmente por uma pequena mudança do código por parte de um programador.

As mudanças de código são normalmente efetuadas, tal como a mudança de credenciais, nomes de ficheiros, entre outros, o que dificulta. Além disso, estas mudanças podem ocultar certos vetores de ataque conhecidos, o que poderá impossibilitar a sua exploração. No entanto, não significa que os mesmos não estejam presentes. Todas as mudanças referidas afetam o processo de *exploitation* presente na aplicação e a mesma tem de ser monitorizada quando se encontra em fase de verificação de um novo alvo.

As potenciais falhas referentes a mudanças que podemos considerar "pessoais" (ou "empresariais") são uma constante no mundo real e embora o funcionamento do programa seja correto, dada à sua automatização e exposição a falhas conhecidas, provavelmente irão existir "falsos positivos" no mundo real devido à falta de caracterização de certos parâmetros consoante o alvo a atacar. Contudo, esta responsabilidade é contida sobre o utilizador da plataforma, visto que o acesso e elaboração dos *scripts* de ataque recaem sobre o mesmo, tendo assim (sobre circunstâncias específicas) que sofrer uma adaptação ao alvo em questão.

Para além disso, há vulnerabilidades que não são facilmente identificáveis pela infraestrutura. É de ressalvar que determinados cenários não apresentam uma representação visual para o atacante do seu sucesso, surgindo assim uma necessidade de verificação manual por parte do utilizador da ferramenta para comprovar este cenário/vetor de ataque para confirmar a sua exploração.

Embora seja uma ferramenta útil, rápida e eficaz, a mesma necessita de ser utilizada sob um registo de monitorização e adaptação constante para conseguir alcançar resultados ótimos no momento das suas análise, precisando assim de uma vertente humana a acompanhar este processo, mesmo possuindo capacidades de automatização, visto que o objetivo é a precisão na deteção de potenciais vulnerabilidades.

6.2 Complementos para a utilização

A ferramenta elaborada avança além dos atuais "scanners de vulnerabilidades" que estão disponíveis, para ser efetuado um teste direto face a uma potencial vulnerabilidade e/ou versão vulnerável de software. Devido a este tipo de abordagem, cada vetor de ataque explorado detém uma pontuação, fazendo com que no final seja atribuída uma pontuação que aglomera todos os pontos anteriormente explorados para cada alvo.

No âmbito de ser alcançada uma classificação mais precisa sobre a "pontuação" obtida,

podem ser utilizadas várias plataformas como "complementos" face ao projeto elaborado, sendo as mesmas(por exemplo): BitSight [56] e SecurityScoreCard [52] para a gestão de risco para o próprio cliente/utilizador.

Os complementos mencionados não são restritos a ferramentas de risco e/ou de classificação, sendo sempre aconselhável a prática da segurança "In-Depth" que se destina à utilização de várias camadas de segurança com o intuito de mitigar possíveis explorações de falhas relativas à ciber-segurança, através da criação de várias barreiras nas diversas camadas tecnológicas presentes na infraestrutura, dificultando assim a atividade maliciosa levada a cabo por agentes maliciosos.

Uma das práticas mundialmente adotadas, consiste na criação de um Security Operations Center (SOC) [54]. O objetivo desta prática consiste na constante análise de todo o tráfego proveniente da internet para os diversos pontos de exposição de uma organização/empresa, com o intuito de ser efetuada uma análise para mitigar possíveis pacotes ou pedidos enviados por agentes maliciosos no momento em que os mesmos possam chegar à rede, evitando possíveis estragos, corrupções de dados, *leaks* de informação entre outras consequências.

A segurança "In-Depth" deve sempre ser utilizada independentemente do cenário em que se encontre qualquer infraestrutura [14], sendo que nenhuma componente do sistema deve ser alvo de apenas um sistema de monitorização ou de testes.

6.3 Conclusão

Os objetivos propostos e delimitados para este projeto enquadravam-se na satisfação de uma análise parcialmente automatizada a vulnerabilidades de clientes/utilizadores finais, através da emulação de um cenário de ataque orquestrado por um agente malicioso, assim como a atribuição de uma pontuação no final após esta análise para uma alerta sobre as potenciais vulnerabilidades informáticas presentes.

Apesar das limitações do projeto, existe uma relação de dependência direta com os objetivos solicitados para a realização do mesmo, ou seja, numa tentativa de encontrar um *standart* para executar todos os testes de segurança sobre uma plataforma única, foi necessário abdicar de precisões a nível da infraestrutura. Deste modo, é possível delegar a responsabilidade da comprovação da correta exploração da vulnerabilidade para o *script* adjacente. Esta delegação não irá ser sempre trivial, ou até mesmo possível em certos casos, sendo necessária uma intervenção manual, devido às diversas formas criativas e em constante evolução que se utilizam hoje em dia no ataque a sistemas informáticos.

O projeto desenvolvido foi de encontro aos objetivos propostos e acordados, sendo utilizada uma automatização parcial dos seus mecanismos e a vertente humana no que diz respeito ao desenvolvimento dos *scripts* utilizados pelo mesmo, numa tentativa de aumentar a taxa de precisão e eliminar resultados "falsos-negativos" e "falsos-positivos" em todo o processo de análise.

Sendo assim, o projeto apresentado tem o potencial de eliminar parte dos vetores de ataque utilizados atualmente, visto assemelhar-se a um "Threat Agent" no seu comportamento e também sobre as ferramentas que possui, no qual se conclui que será uma excelente adição às medidas de segurança de qualquer alvo numa análise periódica atualizada e confiável por parte deste software.

6.4 Trabalho Futuro

A presente infraestrutura encontra-se desenvolvida com camadas de abstração, com o uso de base de dados relacionais, *scripts* efetuados com propriedades modulares, abstenção de linguagens de programação sobre a invocação dos subprocessos, utilização de JSON para o mapeamento das vulnerabilidades, entre outras. Estas escolhas foram efetuadas para facilitar o trabalho futuro sobre a aplicação e a ter em conta funcionalidades futuras para as quais já estão atualmente a congregar informação, como se pode verificar pela existência das tabelas "scoring" e "log_attack" dentro da base de dados relacional. Sendo que estas tabelas poderão ser utilizadas no futuro para a elaboração de estudos estatísticos e também possuem como objetivo facilitar futuras implementações para a visualização de vulnerabilidades.

Para uma devida utilização da aplicação é também necessária a correção das atuais restrições referidas anteriormente contra o mundo real, das quais se destacam na identificação da exploração de vetores de ataque que não possuam uma representação visual de sucesso quanto à eficácia do ataque em questão.

Por último, é necessário manter a base de dados de ataques disponíveis atualizada consoante as possíveis vulnerabilidades nos sistemas alvo e acompanhadas dos CVEs com o maior nível de impacto mais recentes, visto estes serem os alvos primários de utilização por parte dos *"Threat Agents"*. É de realçar que a aplicação necessita de estar sempre a sofrer atualizações ao nível do banco de *scripts* para conseguir ter uma alta taxa de credibilidade, ou seja, exige uma alta taxa de manutenção para poder provar resultados conclusivos e precisos, visto que o mundo da ciber-segurança nunca abranda no que toca à procura de novas vulnerabilidades e vetores de ataque inovadores e nunca antes pensados e/ou vistos.

Bibliografia

[1] Rajika Abeyrathne. *Chidori - Analyzing the impact of Ruby's dynamic features on performance*. PhD thesis, Informatics Institute of Technology, 08 2020.

[2] Charles Anderson. Docker [software engineering]. *Ieee Software*, 32(3):102–c3, 2015.

[3] Jim Anderson. Python bindings: Calling c or c++ from python, 1999.

[4] Nadarajah Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *International Workshop on Security Protocols*, pages 28–41. Springer, 2003.

[5] AVAST. What is a computer exploit? Online, August 2021.

[6] Daniel Bartholomew. Mariadb vs. mysql. *Dostopano*, 7(10):2014, 2012.

[7] Leyla Bilge and Tudor Dumitraş. Before we knew it: an empirical study of zero-day attacks in the real world. *dl.acm.org*, pages 833–844, 2012.

[8] Britannia. Advantages and disadvantages of automation. Online, 2021.

[9] Sung-Do Chi, Jong Sou Park, Ki-Chan Jung, and Jang-Se Lee. Network security modeling and cyber attack simulation methodology. In *Australasian Conference on Information Security and Privacy*, pages 320–333. Springer, 2001.

[10] CISA. Ddos quick guide. Online, 2021.

[11] CISA. Man-in-the-middle. Online, 2021.

[12] CISA. Ransomware. Online, 2021.

[13] CISA. Sql injection. Online, 2021.

[14] Mark Fabro David Kuipers. Control systems cyber security: Defense in depth strategies. Online, July 2021.

[15] Andy Dearden. Ida-s: A conceptual framework for partial automation. In Ann Blandford, Jean Vanderdonckt, and Phil Gray, editors, *People and computers XV : interaction without frontiers*, B C S Conference Series, pages 213–228. Springer, London, 2001. Joint proceedings

of the 15th Annual Conference of the British-HCI-Group (HCI), and the 14th Annual Conference of the Association-Francophone-Interaction-Human-Machine (IHM) Sept. 10-14, (2001) Lille, FRANCE.

[16] Ken Dye. *People and Computers XV—Interaction without Frontiers: Joint Proceedings of HCI 2001 and IHM 2001*. Springer-Verlag London, 1 edition, 2001. ISBN: 978-1-85233-515-1,978-1-4471-0353-0.

[17] FIRST. Cvss metrics table. Online, 2021.

[18] David Flanagan and Yukihiro Matsumoto. *The Ruby Programming Language: Everything You Need to Know*. "O'Reilly Media, Inc.", 2008.

[19] Fortinet. Fortinet top 20 cyber attacks. Online, 2021.

[20] Jacob Fox. Cybersecurity statistics for 2021. Online, 2021.

[21] Robert David Graham. rdpscan by robert david graham. Online, September 2021.

[22] Robert David Graham. Robert graham consultant, errata security. Online, September 2021.

[23] GuardiCore. Infection monkey. Online, September 2021.

[24] Himanshu Gupta and Rohit Kumar. Protection against penetration attacks using metasploit. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions)*, pages 1–4. IEEE, 2015.

[25] Hahwul. Github repository for metasploit-autopwn. Online, September 2021.

[26] Lane Harrison, Riley Spahn, Mike Iannacone, Evan Downing, and John R Goodall. Nv: Nessus vulnerability visualization for the web. In *Proceedings of the ninth international symposium on visualization for cyber security*, pages 25–32, 2012.

[27] Oona A Hathaway, Rebecca Crootof, Philip Levitz, Haley Nix, Aileen Nowlan, William Perdue, and Julia Spiegel. The law of cyber-attack. *California Law Review*, pages 817–885, 2012.

[28] T. Santhanakrishnan Hema Krishnan, M.Sudheep Elayidom. mongodb paper. Online, July 2021.

[29] Anastassia Lauterbach. Cyber-security: Full prevention is not possible. Online, September 2021.

[30] Michael E. Lee. Optimization of computer programs in c. *icps.u-strasbg.fr*, 1(1):1, 2021.

[31] Avijit Mallik. Man-in-the-middle-attack: Understanding in simple words. *researchgate.net*, 12 2020. doi:http://dx.doi.org/10.22373/cj.v2i2.3453.

[32] John Matherly. Shodan.io. Online, September 2021.

[33] Wolfgang McGavran. Intended consequences: regulating cyber attacks. *Tul. J. Tech. & Intell. Prop.*, 12:259, 2009.

[34] MITRE. About cve. Online, August 2021.

[35] MITRE. About cve records. Online, July 2021.

[36] MITRE. Search tips cve. Online, July 2021.

[37] MITRE. About cwe. Online, August 2021.

[38] MITRE. About cwe. Online, July 2021.

[39] MITRE. Common weakness enumeration top 25. Online, 2021.

[40] MySQL. Mysql whitepaper. Online, July 2021.

[41] NIST. Cve-2014-6271 detail. Online, September 2021.

[42] NIST. Cvss. Online, 2021.

[43] NIST. Vulnerability metrics. Online, 2021.

[44] Christine Niyizamwiyitira and Lars Lundberg. Performance evaluation of sql and nosql database management systems in a cluster. *International Journal of Database Management Systems*, 9(6):1–24, 2017.

[45] NullArray. Github repository for autosploit. Online, September 2021.

[46] Ronny Richardson and Max M North. Ransomware: Evolution, mitigation and prevention. *International Management Review*, 13(1):10, 2017.

[47] Dennis M Ritchie, Brian W Kernighan, and Michael E Lesk. *The C programming language*. Prentice Hall Englewood Cliffs, 1988.

[48] Gabriela Roldán-Molina, Mario Almache-Cueva, Carlos Silva-Rabadão, Iryna Yevseyeva, and Vitor Basto-Fernandes. A comparison of cybersecurity risk analysis tools. *Procedia Computer Science*, 121:568–575, 2017. ISSN: 1877-0509. doi:https://doi.org/10.1016/j.procs.2017.11.075.

[49] Diário de Notícias Rui da Rocha Ferreira. O vírus informático que paralisou os hospitais cuf. *Diário de Notícias*, 1(1):1, 2018. doi:10.1364/AO.21.002758.

[50] Michel F Sanner et al. Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61, 1999.

[51] Linux Security. Linux security tools analysis. Online, September 2021.

[52] SecurityScorecard. Securityscorecard. Online, September 2021.

[53] Ryan Spangler. Analysis of remote active operating system fingerprinting tools. *University of Wisconsin - Whitewater*, 1:34, 2003.

[54] Sathya Chandran Sundaramurthy. An anthropological study of security operations centers to improve operational efficiency. Online, July 2021.

[55] Sunguardas. The consequences of a cyber security breach. Online, September 2021.

[56] BitSight Technologies. Bitsight. Online, September 2021.

[57] Tessian. Phishing statistics. Online, July 2021.

[58] Stephen Turner. Security vulnerabilities of the top ten programming languages: C, java, c++, objective-c, c#, php, visual basic, python, perl, and ruby. *Journal of Technology Research*, 5:1, 2014.

[59] Verizon. Data breach investigations report. Online, July 2021.

[60] Verizon. Results and report analysis for 2021. Online, July 2021.

[61] Navjot Verma and Amardeep Kaur. A detailed study on prevention of sqli attacks for web security. *International Journal of Science and Engineering Applications*, 4(4):308–311, 2015.

[62] Anil Baran Yelken. Github repository for arpag. Online, September 2021.