



UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rômulo Augusto Oliveira Cruz Bittencourt de Almeida

**Heuristic phishing detection based on web crawling and user behaviour  
monitoring with a deterministic approach for cybersecurity**

Florianópolis  
2022

Rômulo Augusto Oliveira Cruz Bittencourt de Almeida

**Heuristic phishing detection based on web crawling and user behaviour monitoring with a deterministic approach for cybersecurity**

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Ciências da Computação.

Supervisor:: Prof.(a)Carla Merkle Westphall, Dra.

Florianópolis

2022

Ficha de identificação da obra elaborada pelo autor,  
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Almeida, Rômulo Augusto Oliveira Cruz Bittencourt de  
Heuristic phishing detection based on web crawling and  
user behaviour monitoring with a deterministic approach  
for cybersecurity / Rômulo Augusto Oliveira Cruz  
Bittencourt de Almeida ; orientadora, Carla Merkle  
Westphall, 2022.  
93 p.

Dissertação (mestrado) - Universidade Federal de Santa  
Catarina, Centro Tecnológico, Programa de Pós-Graduação em  
Ciência da Computação, Florianópolis, 2022.

Inclui referências.

1. Ciência da Computação. 2. Phishing. 3. Vetores de  
Ataque. 4. Engenharia Social. 5. Comportamento do Usuário.  
I. Westphall, Carla Merkle. II. Universidade Federal de  
Santa Catarina. Programa de Pós-Graduação em Ciência da  
Computação. III. Título.

Rômulo Augusto Oliveira Cruz Bittencourt de Almeida

**Heuristic phishing detection based on web crawling and user behaviour  
monitoring with a deterministic approach for cybersecurity**

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca  
examinadora composta pelos seguintes membros:

Prof. Eduardo Luzeiro Feitosa, Dr.  
Universidade Federal do Amazonas

Prof. Jean Everson Martina, Dr.  
Universidade Federal de Santa Catarina

Prof. Maicon Rafael Zatelli, Dr.  
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi  
julgado adequado para obtenção do título de mestre em Ciências da Computação.

---

Patricia Della Méa Plentz  
Coordenação do Programa de  
Pós-Graduação

---

Prof.(a) Carla Merkle Westphall, Dra.  
Supervisor:

Florianópolis, 2022.

This work is dedicated to my mother, Laura Cruz, and everyone who has positively and negatively impacted my journey.

## **ACKNOWLEDGEMENTS**

I thank my mother, Laura Cruz, and my family for their constant support and encouragement in my life. I also thank my wife Ana Collares for her support during my Master's period. I thank Professor Carla Westphall for guiding me during the preparation of this work. All LRG members and colleagues contributed in some way to this research. Finally, I would like to thank professors Jean Martina, Eduardo Feitosa, and Maicon Zatelli while integrating my dissertation committee.

*“All we have to decide is what to do with the time that is given us.”*  
*(J.R.R. Tolkien, The Fellowship of the Ring, 1937)*

## RESUMO

Com o aprimoramento das técnicas utilizadas pelos atacantes, a detecção e prevenção de ameaças como *phishing* e *malware* podem representar um problema e desafio computacional. Uma pesquisa recente descobriu que o *phishing* e a infecção por código malicioso são as principais ameaças desencadeadas pela engenharia social. Neste trabalho, são analisados os vetores de ataque que causam essas ameaças, propondo um método de verificação de *strings* específicas em URLs e mensagens de e-mail, que pode ser usado em conjunto com proxies e filtros anti-spam. O método foi implementado em um cenário experimental e é capaz de detectar a presença dos principais elementos que têm contato direto com o usuário, tais como: campos de formulário, redirecionamento de *links* e arquivos para *download*. O método proposto foi capaz de detectar URLs de phishing em tempo real, com precisão de 97,66% e tempo médio de 30 segundos. Além disso, um novo método de mapeamento do comportamento do usuário é proposto neste trabalho. Por meio deste, é possível demonstrar que os vetores de ataque utilizados em cada etapa da exploração apresentam sempre o mesmo comportamento, seguindo uma sequência de etapas já conhecida. A estrutura de mapeamento pode ser representada por meio de uma máquina de estados finitos (DFA), onde cada estágio de interação do usuário com um possível vetor de ataque representa um ponto determinístico da máquina finita. Os resultados obtidos através desta implementação podem ser utilizados na detecção de manobras maliciosas em tempo real e na conscientização de usuários em segurança da informação.

**Palavras-chave:** Phishing. Vetores de Ataque. Engenharia Social. Comportamento do Usuário.



## RESUMO EXPANDIDO

### Introdução

Na literatura, o *phishing* pode ser definido pela criação de páginas falsas na WEB, onde os criminosos disparam *e-mails* em massa com os *links* para estas páginas. Nestas mensagens os agentes maliciosos fingem ser outros indivíduos ou organizações aparentemente legítimas, com o intuito de induzir os usuários a fornecer dados confidenciais. De acordo com (KOYUN; AL JANABI, 2017), os ataques de *phishing* também podem ser classificados em: *spear phishing*, *whaling phishing* e *phishing* de comprometimento de *e-mail* corporativo.

Ataques de *spear phishing* referem-se a manobras direcionadas a indivíduos ou grupos específicos, usando seus nomes para fazer reivindicações ou comunicações falsas. Estas manobras exigem a coleta de informações *online* sobre a vítima usando técnicas de engenharia social e espionagem digital. Essas informações geralmente são coletadas através de varreduras em redes sociais mas também podem ser realizadas através de softwares como o SET (*Social Engineering Toolkit*) no *Kali Linux* (KOYUN; AL JANABI, 2017).

O *whaling phishing* é também um *spear phishing* mas além disso visa indivíduos em altos cargos em empresas. Por fim, o *phishing* de comprometimento de *e-mail* corporativo é similar ao *whaling phishing* com a diferença de que seu intuito final é obter acesso a caixa de mensagens e outras informações privadas. Neste tipo de ataque as técnicas de engenharia social são combinadas com a disseminação de *malware* através do envio de anexos contendo *trojans*, *keyloggers* e outros *malwares*. Para a contextualização deste trabalho podemos destacar três vetores de ataque presentes nas categorias de *phishing* descritas anteriormente, são eles: divulgação de páginas falsas e fraudulentas, envio de mensagens de *e-mail* maliciosas e a disseminação de *malware* através de conteúdo de *download*.

Ameaças como o *phishing* e a infecção por *malware* representam as principais manobras da engenharia social para atingir usuários (THAKUR; PATHAN, 2020) e na grande maioria dos casos o indivíduo não está conscientizado para reagir a estas ameaças. Os ataques de *phishing* são caracterizados comumente pelo envio de *e-mails* e páginas maliciosas por parte dos criminosos, contendo informações aparentemente legítimas, induzindo as vítimas a fornecerem dados confidenciais como senhas e dados de cartão de crédito (ABUZURAIQ et al., 2020), ou até mesmo últimas informações sobre o COVID-19. Agentes maliciosos também estão usando nomes relacionados ao coronavírus e a entidades governamentais nos títulos de arquivos maliciosos, na

tentativa de induzir os usuários a abri-los. Estes arquivos na verdade estão infectados com aplicações maliciosas como *ransomwares* e *keyloggers* (AHMAD, 2020).

Atualmente diversas pesquisas investigam técnicas *anti-phishing*, podemos dividir as técnicas utilizadas em duas categorias: técnicas baseadas em lista e técnicas heurísticas (RAO; PAIS, 2019). A proposta de análise de URLs e mensagens de *e-mail* apresentada neste trabalho se encontra entre as duas categorias, fazendo uso de técnicas baseadas em lista e técnicas heurísticas. O método utilizado contribui para detecção e extração de elementos característicos de *phishing*, podendo ser incorporada em cenários de segurança comumente encontrados.

## Objetivos

Este trabalho tem como objetivo desenvolver um método heurístico para detecção de *phishing* baseado em *web crawling*. Os módulos propostos devem detectar os elementos que caracterizam a ameaça em tempo real, evitando assim a ocorrência de fraudes. Além disso, este trabalho também contribui para o desenvolvimento de um método de monitoramento do comportamento do usuário, capaz de mapear as interações do usuário com o *phishing* e infecção por *malware*, por meio de sensores de monitoramento baseados em um automato finito determinístico (DFA). Os objetivos específicos são:

- Desenvolver um método comparativo de bancos de dados de phishing catalogados, capaz de detectar se a URL de destino já foi relatada como phishing no banco de dados utilizado.
- Desenvolver um algoritmo de detecção de phishing capaz de detectar os elementos característicos da ameaça nas páginas acessadas pelos usuários e em tempo real.
- Desenvolver uma métrica de avaliação de páginas de phishing que possa ser utilizada para detectar páginas fraudulentas em conjunto com o algoritmo proposto.
- Validar o método de detecção de phishing em um cenário experimental controlado passível de reprodução.

## Metodologia

Este trabalho foi realizado através de uma metodologia quantitativa, de natureza aplicada, elaborada através de uma pesquisa exploratória contendo revisão da bibliografia e experimentação em um cenário controlado. Para isso, foram levantadas soluções da literatura envolvendo a detecção de *phishing* em tempo real e bases de catalogação de *phishing*. Além disso, foram levantados trabalhos envolvendo o mapeamento

do comportamento computacional dos usuários. Esses trabalhos foram avaliados nos seguintes aspectos: tipo de amostras, bases de catalogação utilizadas, método de detecção aplicado, independência de idioma, tipo de análise (tempo real ou *post mortem*) e tipo de implementação computacional. Em seguida, um método de detecção de páginas de *phishing* em tempo real foi projetado, contendo uma métrica de avaliação de páginas maliciosas baseada em índices de incidência e criticidade dos vetores de ataque presentes nas URLs. Uma análise acerca da performance do método de *web crawling* para detecção de elementos que compoem o *phishing* foi realizada, com o propósito de investigar a viabilidade de sua utilização em um cenário real. Por fim, um método de detecção as ações do usuário foi acrescentado ao modelo, com o intuito de detectar em tempo real o momento em que o usuário interage com os vetores de ataque mapeados através do algoritmo de detecção de *phishing*. O experimento foi executado em ambiente virtualizado, baseado em ferramentas *open source* e similar aos cenários corporativos comumente encontrados. Com base nos dados encontrados nos experimentos, alguns resultados foram destacados.

## Resultados e Discussão

Os principais resultados retirados dos experimentos e demais análises estão descritos a seguir:

- Inicialmente, para a criação e preenchimento do banco de dados, foi utilizada a importação do banco de dados *PhishTank* no formato csv. A escolha da base *Phishtank* entre outras plataformas se deu devido à disponibilidade de acesso aos registros da plataforma e à possibilidade de conexão através de sua API para realização de consultas à base.
- Aproximadamente 18.000 amostras de URL foram avaliadas, onde aproximadamente apenas 12.000 URLs puderam ser removidas do banco de dados *Phish-Tank* e avaliadas durante o período, pois, um conjunto aproximado de 6.000 URLs não estava mais ativo ou disponível para acesso HTTP/HTTPS. Ao final, o conjunto de amostras avaliadas representou um total de 12.350 URLs válidas, cerca de 12.000 URLs retiradas diretamente da base do *Phishtank* (contando atualizações a cada hora), e outras 350 novas URLs detectadas diretamente pelo método proposto neste trabalho.
- O algoritmo proposto foi implementado por meio da API Collector (descrita no capítulo quatro) e atingiu uma precisão de 97,66% na detecção de páginas de *phishing*. Para as URLs coletadas diretamente da *PhishTank Database* a acurácia foi de 84.81%.
- Ao final do monitoramento de mensagens maliciosas nos *e-mails*, aproxima-

mente 70% foram classificados como *spear phishing* e 30 % como outros tipos de *phishing*. Em muitos *e-mails*, os invasores simularam *e-mails* válidos de outras instituições ou funcionários da empresa solicitando informações financeiras ou contratuais específicas.

- Através da função de cálculo de incidência implementada, foi possível detectar mensagens repetidas de *spear-phishing* e outros tipos de *phishing* na caixa de entrada dos usuários. De um modo geral, as mensagens maliciosas que atingiram a taxa máxima de incidência (5+) foram mensagens relacionadas ao COVID-19 e *links* falsos de ajuda de emergência do governo.
- As principais URLs maliciosas estavam vinculadas aos seguintes temas: COVID-19, ajuda e serviços governamentais, instituições financeiras, serviços de compra e venda (*e-commerce*) e serviços de passagens aéreas. Além disso, aproximadamente 81% das URLs não possuíam certificado digital e, em mais de 80% dos casos, tinham o ataque homográfico como vetor de ataque para confundir os usuários.
- Levando em conta as 80 URLs de *phishing* detectados, a combinação dos vetores de ataque *Homographic Attack (HA)*, *Certificate Missing (CM)* e *Form Fields (FF)* apareceram juntos em 57 URLs. Com isso, 71,25% das mensagens de *phishing* de novas URLs detectadas apresentavam este recurso.

### **Considerações Finais**

Este trabalho propôs e implementou um método de detecção de *phishing* em tempo real, com a validação do mesmo em um cenário experimental controlado. Em complemento ao método de detecção de *phishing* apresentado, foram implementadas maneiras computacionalmente viáveis de monitorar as ações do usuário. Os resultados obtidos permitiram observar que o Banco de Dados *PhishTank* não teve um papel relevante no monitoramento e notificação de páginas de *phishing* no cenário brasileiro. Novas investigações podem ser realizadas sobre a eficiência da plataforma em relação à linguagem PT-BR e ao cenário brasileiro de URLs de *phishing*. Este trabalho é independente de linguagem e de plataforma para avaliação de *phishing* comparado aos trabalhos relacionados. Além disso, seu principal destaque em relação aos outros trabalhos avaliados está na detecção em tempo real das páginas maliciosas, no exato momento em que são acessadas pelo usuário. O algoritmo de detecção é flexível, possibilitando assim adaptar novos vetores de ataque aos motores de busca.

A técnica de *Web Crawling* proposta neste trabalho apresenta algumas limitações na extração de informações de páginas com estruturas mais complexas, que utilizam outras estruturas de código embutidas no *layout* (*JavaScript*, *JSON*, entre outras).

Páginas maiores e mais pesadas levaram mais de 5 minutos para retornar os valores extraídos, especialmente ao verificar *hiperlinks* de redirecionamento implementados por meio de chamadas de código nas páginas. Pesquisas futuras devem aprimorar o algoritmo e os métodos de busca utilizados.

Em adição ao método de detecção de *phishing* os módulos de captura do comportamento do usuário apresentaram bons resultados, detectando ações como cliques e interações em elementos da página, abertura, fechamento de abas, preenchimento de campos de formulários e *download* de arquivos. A taxonomia proposta desempenhou papel fundamental no desenvolvimento desse método de monitoramento. Através do mapeamento exato das atividades, foi possível determinar o momento exato em que um usuário entrou em contato com uma manobra maliciosa e se a manobra foi implementada ou não. Esse método de extração apresentou problemas relacionados ao tempo de processamento das atividades no navegador. Algumas ações demoraram mais de cinco minutos para serem catalogadas, dependendo diretamente dos recursos de memória disponíveis. Futuramente, serão investigadas formas mais viáveis computacionalmente de capturar ações do usuário através do navegador para que não realizem alto consumo de recursos computacionais.

**Palavras-chave:** Phishing. Vetores de Ataque. Engenharia Social. Comportamento do Usuário.

## ABSTRACT

With the improvement of the techniques used by attackers, the detection and prevention of threats such as phishing and malware can represent a problem and computational challenge. A recent survey found that phishing and infection by malicious code are the top threats triggered by social engineering. In this work, the attack vectors that cause these threats are analyzed, proposing a method of verifying specific strings in URLs and e-mail messages, which can be used in conjunction with proxies and anti-spam filters. The method was implemented in an experimental scenario and is capable of detecting the presence of the main elements that have direct contact with the user, such as form fields, redirection of links and files to download. The proposed method was able to detect phishing URLs in real-time, with an accuracy of 97.66% and an average time of 30 seconds. Furthermore, a new user behavior mapping method is proposed in this work. Through this, it is possible to demonstrate that the attack vectors used in each stage of the exploitation always present the same behavior, following a sequence of steps already known. The mapping structure can be represented by a finite state machine (DFA), where each user interaction stage with a possible attack vector represents a deterministic point of the finite machine. The results obtained through this implementation can be used in the detection of malicious maneuvers in real-time and in the awareness of users in information security.

**Keywords:** Phishing. Attack Vectors. Social Engineering. User Behaviour.

## LIST OF FIGURES

Figure 1 – Classification of Social Engineering Attacks (SALAHDINE; KAABOUC, 2019) . . . . .	23
Figure 2 – Example of creating phishing pages and fraudulent emails (JAMES, Lance, 2005) . . . . .	24
Figure 3 – Illustration of phishing experiment (JAGATICet al., 2007) . . . . .	25
Figure 4 – Types of ransomware (ANDRONIOet al., 2015) . . . . .	27
Figure 5 – The typical steps used by ransomware to encrypt and decrypt a user's data. (BEAMANet al., 2021) . . . . .	29
Figure 6 – Server-side web threats (SADQI; MALEH, 2021) . . . . .	30
Figure 7 – Taxonomy of social engineering attacks and threats (ALDAWOOD;SKINNER, 2020) . . . . .	31
Figure 8 – Exemple of Finite Automaton M4 -(SIPSER, 1996) . . . . .	32
Figure 9 – Exemple of Crawler Architecture - (OLSTON; NAJORK,2010) . . . . .	33
Figure 10 – Phishing Detection - Comparative Table of Works Related to this Work	37
Figure 11 – User Behavior - Comparative Table of Works Related to this Work .	41
Figure 12 – Experimental Scenario . . . . .	43
Figure 13 – Virtual Machines and Configurations . . . . .	44
Figure 14 – Connections between python API and Phishtank Database . . . . .	45
Figure 15 – Database Collections . . . . .	46
Figure 16 – Example of url-databse collection . . . . .	48
Figure 17 – e-mail collection example . . . . .	50
Figure 18 – Table of Metrics - Classification of Attack Vectors and Score . . . . .	51
Figure 19 – Python API - Scraping Engines . . . . .	54
Figure 20 – Attack vector incidence percentage . . . . .	55
Figure 21 – Attack vector incidence percentage . . . . .	57
Figure 22 – API execution and collections interactions . . . . .	59
Figure 23 – Taxonomy of Attack Vectors and User Actions . . . . .	62
Figure 24 – New APIs and Additions . . . . .	66
Figure 25 – User Virtual Machine Simulation . . . . .	67
Figure 26 – DFA state diagram . . . . .	70
Figure 27 – Examples of User Maturity Tree . . . . .	74
Figure 28 – Phishing Detection Accuracy - Hit/Miss . . . . .	75
Figure 29 – Phishing Detection Accuracy - New Samples Vs PhishTank Database	76
Figure 30 – Phishing messages received via mailbox . . . . .	77
Figure 31 – Phishing messages received via mailbox - Pt2 . . . . .	78
Figure 32 – Phishing occurrences . . . . .	80

## LIST OF ABBREVIATIONS AND ACRONYMS

AES	Advanced Encryption Standard
AJAX	asynchronous JavaScript and XML
API	application programming interface
C2	command and control
CDATA	character data
COVID-19	SARS-CoV-2 Virus Disease
CPS	cyber-physical systems
CSS	cascading style sheets
csv	Character-separated values
DFA	Deterministic Finite Automaton
DL	deep learning
docx	Microsoft Word document
EVSDT	equal variance sign detection theory
HMAC-MD5	hash-based message authentication code with message digest algorithm 5
HTML	HyperText Markup Language
HTTP	hypertext transfer protocol
HTTPS	hypertext transfer protocol secure
ID	identity
IoT	internet of things
JSON	JavaScript object notation
LDAP	Lightweight Directory Access Protocol
NIST	National Institute of Standards and Technology
NLP	natural language processing
NoSQL	Not Only SQL
PC	personal computer's
pdf	portable document format
PSHCS	phishing URLs hosted on compromised servers
RaaS	ransomware as a service
RSA	Rivest-Shamir-Adleman
SET	social engineering toolkit
SMS	Short Message Service
SOM	self-organization map
TDF	theoretical domain framework
UMBC	university of Maryland
URL	Uniform Resource Locator
UVSDT	unequal variance SDT



WEB	world wide web
XML	extensible markup language
zero-day	newly created or unknown

## LIST OF SYMBOLS

$\Sigma$	Input Alphabet
$\delta$	transition rules
$\epsilon$	Empty Entry

## CONTENTS

<b>1</b>	<b>INTRODUCTION</b> . . . . .	<b>19</b>
1.1	RESEARCH PROBLEM AND MOTIVATION . . . . .	20
1.2	OBJECTIVES . . . . .	21
<b>1.2.1</b>	<b>General Objectives</b> . . . . .	<b>21</b>
<b>1.2.2</b>	<b>Specific Objectives</b> . . . . .	<b>21</b>
1.3	WORK STRUCTURE . . . . .	21
<b>2</b>	<b>BASIC CONCEPTS</b> . . . . .	<b>22</b>
2.1	SOCIAL ENGINEERING AND HUMAN FACTOR . . . . .	22
2.2	PHISHING . . . . .	23
2.3	MALWARE . . . . .	26
2.4	TAXONOMY IN CYBERSECURITY . . . . .	28
2.5	AUTOMATA THEORY - DETERMINISTIC FINITE AUTOMATON (DFA)	30
2.6	WEB CRAWLING . . . . .	32
<b>3</b>	<b>RELATED WORKS</b> . . . . .	<b>34</b>
3.1	SELECTED WORKS AND SYSTEMATIC REVIEW . . . . .	34
3.2	PHISHING DETECTION . . . . .	35
3.3	USER BEHAVIOR . . . . .	38
<b>4</b>	<b>PHISHING DETECTION AND USER BEHAVIOUR MAPPING</b> . . . .	<b>42</b>
4.1	HEURISTIC PHISHING DETECTION . . . . .	42
<b>4.1.1</b>	<b>Implementation and experimental scenario</b> . . . . .	<b>43</b>
<b>4.1.2</b>	<b>Collections</b> . . . . .	<b>45</b>
<b>4.1.3</b>	<b>URL score and rating metrics</b> . . . . .	<b>55</b>
<b>4.1.4</b>	<b>API Execution and Collections Interactions</b> . . . . .	<b>57</b>
4.2	USER BEHAVIOR MAPPING . . . . .	60
<b>4.2.1</b>	<b>Taxonomy of User Actions and Attack Vectors</b> . . . . .	<b>60</b>
<b>4.2.2</b>	<b>Additional Implementations in Experimental Scenario and new API modules</b> . . . . .	<b>65</b>
<b>4.2.3</b>	<b>Finite State Machine Implementation</b> . . . . .	<b>69</b>
<b>4.2.4</b>	<b>Maturity trees and user profile</b> . . . . .	<b>72</b>
<b>5</b>	<b>EXPERIMENTAL RESULTS</b> . . . . .	<b>75</b>
<b>6</b>	<b>CONCLUSIONS</b> . . . . .	<b>83</b>
	<b>REFERENCES</b> . . . . .	<b>87</b>

## 1 INTRODUCTION

The digital medium has become vital for the exchange of information and data traffic. It has enabled flexibility and ease in accessing information, and alongside this evolution, there has been an increase in the number of users, resulting in the exponential growth of digital crimes (BARRETT, 2018). It is worth highlighting the atypical period that the world is going through due to the SARS-CoV-2 Virus Disease (COVID-19) pandemic, where cybercriminals are taking maximum advantage to reach the most significant number of users. Malicious activities have increased dramatically due to the side effects of working online from home, coinciding with social distance measures needed to face the pandemic, causing a higher number of information security incidents. Cybersecurity's importance goes beyond the individual concern of each user; it must be part of each process in organizational settings and can be seen as a growing issue of public concern.

The last few years have been marked by several incidents. These occurrences are responsible for an approximate loss of forty-two billion dollars a year, reaching up to a hundred billion dollars, when taking into account the unreported cases (LAVION et al., 2020). The survey of (LAVION et al., 2020) describes that of the more than five thousand correspondent companies, 47% experienced fraud or security incidents within the last 24 months. In addition, it has been observed that attackers have focused their efforts on organizations involved in combating the pandemic, such as hospitals, manufacturers of medical products and supplies. Also, in the year 2020, the main form of attack was through infection by ransomware through vulnerabilities in office tools, phishing and redirection pages (GOSTEV, 2020).

Threats such as phishing and infection by malware represent the main maneuvers of social engineering to reach users (THAKUR; PATHAN, 2020). In the vast majority of cases, the individual is not even aware of how to react to these threats. The human factor is explored through social engineering, which can be seen as one of the most efficient ways to carry out a malicious action (HADLINGTON, 2017). Accordingly, it is possible to observe the user as the main source of vulnerabilities in a computational scope, such as phishing. Another point to be observed in the study of the social engineering action on the user is the user itself, as in Thakur and Pathan 2020. It is necessary to develop efficient ways to map the user's computational behavior when interacting with these threats.

Phishing attacks are commonly characterized by the sending of e-mails and malicious pages by criminals, containing legitimate information, thereupon inducing victims to provide confidential data such as passwords and credit card data (ABUZURAIQ et al., 2020), or even the latest information on COVID-19. Malicious agents are also using names related to coronavirus and government entities in the titles of malicious files, in

an attempt to trick users into opening them. These files are infected with malicious applications, for instance, ransomware and keyloggers (AHMAD, 2020). Observing these factors, it is inevitable to have a minimum security method in a computational environment. To promote such a level of protection, the most diverse security mechanisms are developed and used. However, even with all these developments, they still do not have significant results in terms of protection of the human factor (ALHARTHI et al., 2020).

From this analysis, the human gap can be seen as the primary breach of computational security. Forging trustworthiness in phishing maneuvers represent the most efficient way to perform a malicious action. This factor highlights the hypothesis that the detection of fake pages is the best strategy to combat phishing; it is essential to find a solution that can efficiently detect these pages (JAIN; GUPTA, 2018). Currently, several types of research investigate anti-phishing techniques, and the techniques used can be divided into two categories: list-based techniques and heuristic techniques (RAO; PAIS, 2019). The proposal for analyzing Uniform Resource Locator (URL) and e-mail messages presented in this work uses both list-based and heuristic techniques.

The method used contributes to detecting and extracting elements characteristic of phishing and can be incorporated into commonly found security scenarios. These detection methods were presented in (ALMEIDA; WESTPHALL, 2020) and will be discussed in detail in this paper. Furthermore, the same scenario used in the article will also be used as a basis for developing a structure for monitoring the user's computational behavior. The proposed methodology has as a principle the mapping of specific attack vectors present in phishing and malicious code infection. For this, a finite state machine (finite automaton) capable of detecting the user's actions when interacting with these threats will be created.

## 1.1 RESEARCH PROBLEM AND MOTIVATION

The exponential growth in the number of frauds involving phishing in recent years can be highlighted as the main motivation for this work. In addition, the difficulty in detecting fraudulent maneuvers in real-time becomes one of the main problems detected on the subject. In current times, mitigating phishing with just blacklists and trusted domains is no longer an effective way to combat the threat.

Efficient real-time Phishing detection is not such a simple task, since, in addition to depending on the available computing resources, it also depends on the user's information security maturity level. Thus, it is a two-way street, a technological revolution in the means of detection and user awareness. In addition, it is also necessary to understand and map the user's meeting point with threats coming from social engineering. Little attention has been paid on the study of user computational actions related to phishing. This topic can be considered as the second main motivation for carrying out this work, the scarcity of research that implements computational methods capable of

detecting user actions when they interact with social engineering threats.

## 1.2 OBJECTIVES

### 1.2.1 General Objectives

This work aims to develop a heuristic method for phishing detection based on web crawling. The proposed modules must detect the elements that characterize the threat in real-time, thus preventing the occurrence of fraud. In addition, this work also contributes to the development of a user behavior monitoring method capable of mapping user interactions with social engineering threats, such as phishing and malware infection, through monitoring sensors based on a machine. of finite states. The framework must map and collect user computational actions during interaction with threats analyzed through browser elements.

### 1.2.2 Specific Objectives

1. Develop a comparative method of catalogued phishing databases capable of detecting whether the target URL has already been reported as phishing in the database used.
2. Develop a phishing detection algorithm capable of detecting the characteristic elements of the threat on pages accessed by users and in real-time.
3. Develop a phishing page evaluation metric that can be used to detect fraudulent pages in conjunction with the proposed algorithm.
4. Validate the phishing detection method in a controlled experimental scenario.

## 1.3 WORK STRUCTURE

The structure of this academic study is organized in such a way as to facilitate the flow of reading and understanding the concepts presented. Chapter 2 contains the basic Concepts. Chapter 3 contains the related works alongside the investigated state-of-the-art themes. The chapter is divided into three sections, where the first section contains a brief description of how the systematic review of the literature was carried out. Second section contains the jobs related to phishing detection. Subsequently, the third section comprises the jobs related to mapping user computational behavior. At the end of each subsections a table of related works are exhibited containing the main topics of divergence between the studies and this research, as well as other important points. Chapter 4 presents the phishing detection method proposed in this work in conjunction with the additional contribution of research on mapping user computational behavior. Chapter 5 expounds the results obtained in this investigation, and finally, Chapter 6 submits the conclusions and final considerations about the work and future directions.

## 2 BASIC CONCEPTS

This chapter contains the bibliographical review regarding the state-of-the-art themes in connection with this work. The sections present the basic concepts involving the topics investigated in the following order: Social Engineering and Human Factor, Phishing, Malware, Taxonomy in Cybersecurity, Automata Theory - Deterministic Finite Automaton (DFA) and finally the web crawling theme.

### 2.1 SOCIAL ENGINEERING AND HUMAN FACTOR

The human factor constitutes one of the primary gaps in information security, which can be exploited through social engineering. In 2002, Kevin Mitnick (MITNICK; SIMON, 2003) described the first expressive concept of social engineering: psychological manipulation of people to disseminate information or perform an action. Rather than directing attacks on systems, social engineers aim to persuade users, manipulating them during the performance of malicious activities such as fraud, access, and misuse of information (YASIN et al., 2019).

Observing the current scenario of security solutions, it is apparent that protection measures have become ineffective against this type of attack (CARLTON et al., 2019). In (HADLINGTON, 2017) a relationship between risky behavior of users and cybersecurity in business environments was traced, analyzing factors such as actions performed on the Internet and impulsiveness. In this examination, 538 UK participants completed an online questionnaire, with 515 responses being used in data analysis. The results demonstrated that Internet addiction and impulsiveness were significant indicators of cybersecurity risk behavior. A positive attitude towards cybersecurity in business was negatively related to risky behavior in cybersecurity. Lastly, the impulsivity measure revealed that attention and motor impulsivity were both previous predictors configured according to cybersecurity risk.

Further noteworthy points were highlighted in (SALAH DINE; KAABOUCH, 2019), where the study suggested some classifications regarding the most common types and variations of social engineering methods. The authors emphasized that access can be classified into several categories, contingent on various possibilities. They can classify these into two categories according to the entity involved: human or software. These categories can also be classified according to how the attack is conducted; through manual means or technical means. Additionally, the two main classifications can be highlighted as direct and indirect. Attacks in the first category use direct contacts between the attacker and the victim in order to perform the attack.

These competencies are undertaken by actions performed through physical contact, eye contact, or voice interactions. They may also require an attacker's presence on the victim's desktop to conduct the attack. Examples include physical access, shoulder

surfing, trashing, diving, phone social engineering, pretense, identity spoofing on tech support calls, and theft of important documents. Finally, through indirect contact, the attacker's carry out an attack, which can be launched remotely through malware software transported by email attachments or Short Message Service (SMS) messages. Examples include phishing, fake software, PopUp windows, ransomware, SMSishing, online social engineering, and reverse social engineering. The classifications presented can be better visualized in figure 1, composed of images taken from (SALAHDINE; KAABOUC, 2019)

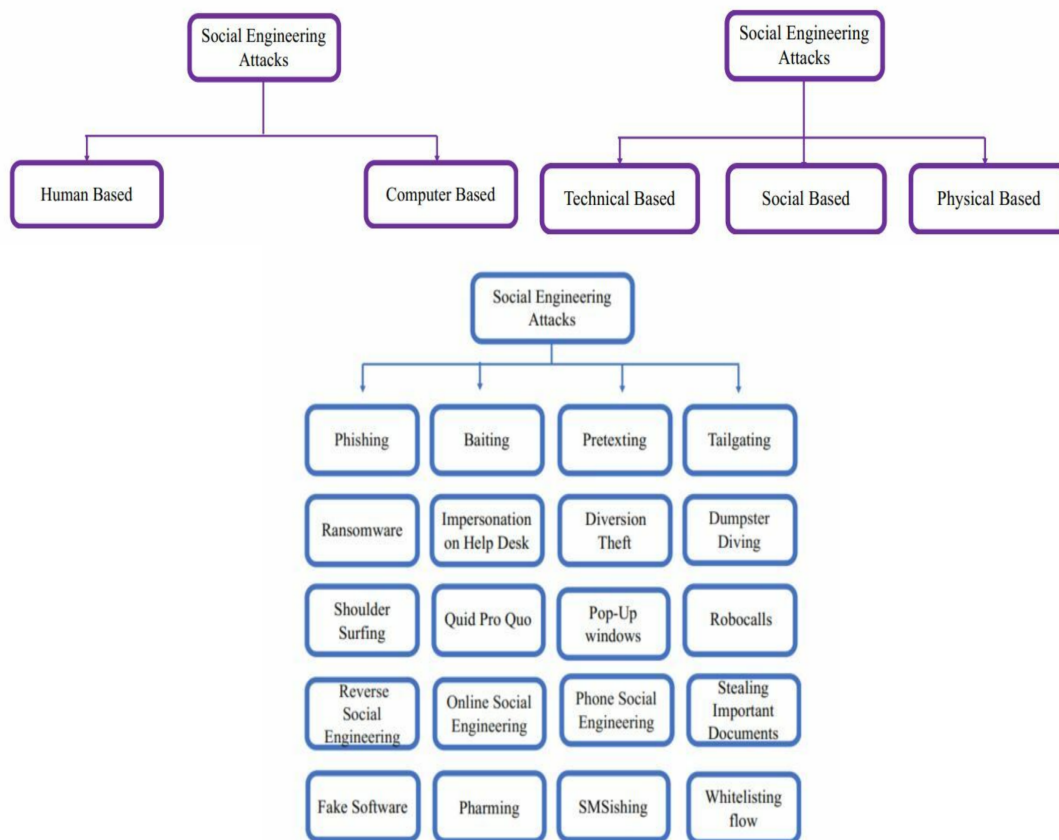


Figure 1 – Classification of Social Engineering Attacks (SALAHDINE; KAABOUC, 2019)

## 2.2 PHISHING

The first approaches to phishing began to emerge in the early 2000s, highlighting the works (JAMES, Lance, 2005), (DHAMIJA et al., 2006) and (JAGATIC et al., 2007). These studies put forward complete approaches and definitions on the subject, demonstrating the steps taken by attackers in carrying out attacks. In the book entitled "Phishing Exposed", (JAMES, Lance, 2005) the concept of Phishing is set forth as a



type of attack in which criminals use spoofed e-mails and fraudulent websites to trick people into providing credentials and/or confidential information. In these messages or false pages, malicious agents intend to be other legitimate individuals or organizations to trick users into providing confidential data. Increasingly sophisticated attacks not only spoof e-mails and websites but can also spoof parts of a user's browser. In this work, experiments were also conducted to demonstrate the creation of phishing pages, servers and the sending of malicious e-mails commonly used in attacks (figure 2).

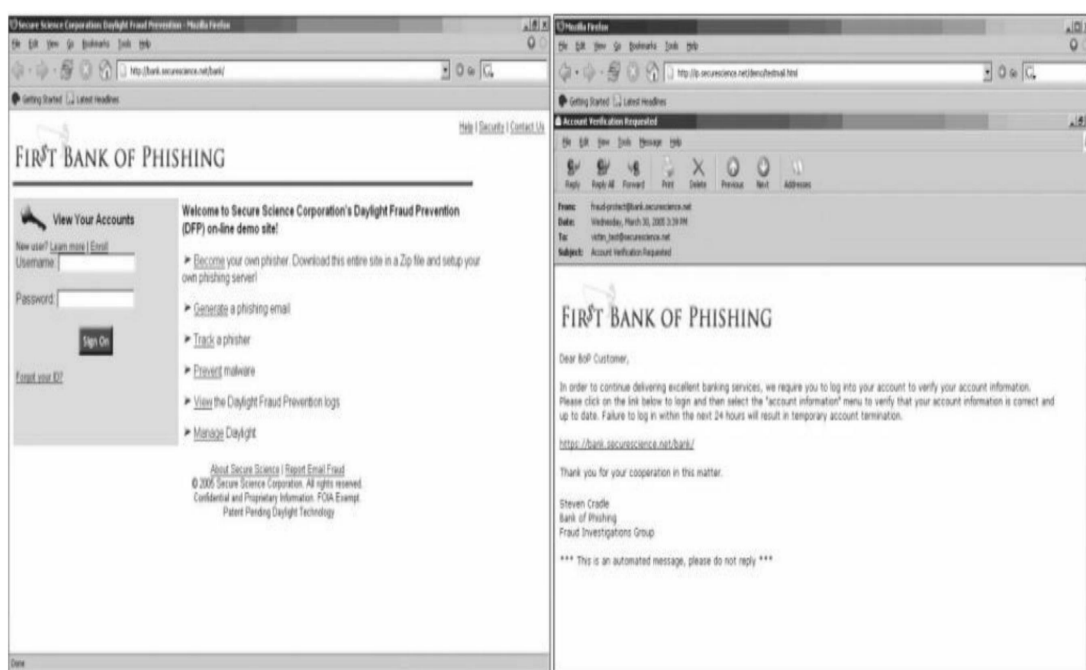


Figure 2 – Example of creating phishing pages and fraudulent emails (JAMES, Lance, 2005)

In a classic study entitled "Why Phishing Works" (DHAMIJA et al., 2006), the authors provided the first empirical evidence about which malicious strategies are effective in deceiving users in general. Firstly, a substantial set of phishing attacks were captured, and thereupon, a set of hypotheses were developed about why these strategies might work. The researchers then evaluated these hypotheses with a usability study, in which twenty sites were presented to twenty-two participants and asked to determine which were fraudulent. As a result, 23% of respondents did not look at the browser considering important phishing detection tips, such as the address bar URL, the presence of a digital certificate, and hypertext transfer protocol secure (HTTPS). Within this context, 40% of the time, the choices were incorrect. The primary discovery factor was that visually deceiving the user is successful in phishing ploys.

In the work "Social Phishing" (JAGATIC et al., 2007) experiments were also conducted to reproduce a phishing attack on individuals affiliated with Indiana University

(figure 3). According to the steps taken, information was first collected on social networks and other public data. The data was then correlated and stored in a relational database where heuristics were used to create e-mail messages spoofed by Eve "as Alice" to Bob (a friend). The message is sent to Bob, and Bob follows the link contained in the e-mail message where an unverified redirect is performed, taking Bob to the attacker's website whuffo.com. The page then asks Bob for his University Credentials, and Bob's credentials are verified with the University authenticator, thus successfully completing the phishing attack. If Bob is not caught, new attempts will be made. The experiment achieved a success rate of up to 72% in Phishing attacks.

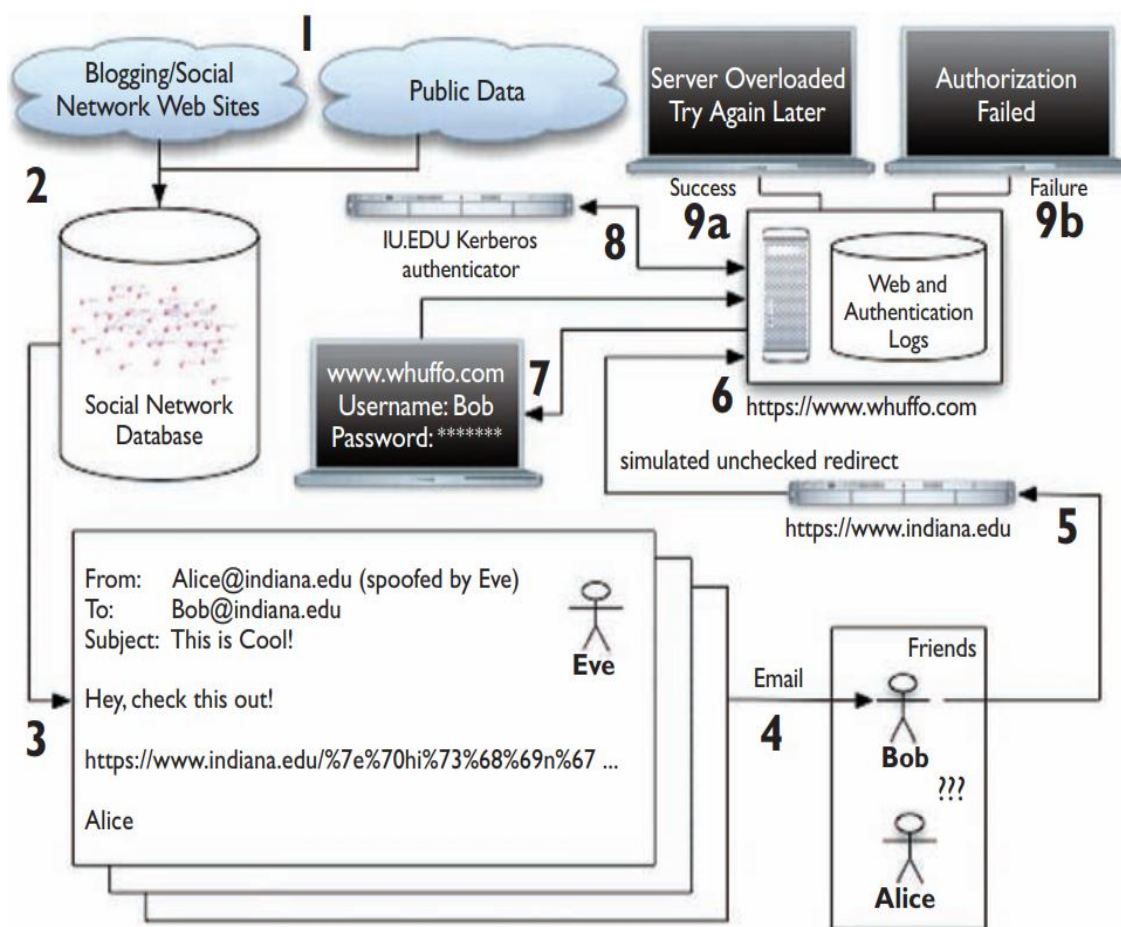


Figure 3 – Illustration of phishing experiment (JAGATICet al., 2007)

Other important aspects can be observed in DaSilva et al. (SILVA et al., 2020) where a phishing timeline was drawn, revealing the emergence of and concerns toward the problem. Through a solid literature review conducted in ScienceDirect Digital Library, it was observed that in the late 1990s, the term phishing was first used to describe a fraudulent scheme. Studies realized between 1999 to 2003 highlighted that criminal organizations used phishing to target banking institutions. In 2004, some studies reported that the practice had become common in the world scenario and scientific research returned to explain the phenomenon. By 2005, it was possible to visualize research

with anti-phishing practices, and shortly after in 2006, the combination of phishing and malware was already observed within the same scope of the attack. In 2008, some investigations indicated computational intelligence as an ally in real-time fraud detection. Finally, from the last topic of the 2008 timeline and onward, proposals based on behavior patterns have consolidated and become a trend in the literature.

Over the years, new phishing concepts and classifications have emerged. According to (KOYUN; AL JANABI, 2017), phishing attacks can further be classified into spear phishing, whaling phishing, and corporate email compromise phishing. Spear phishing attacks refer to maneuvers targeting specific individuals or groups, using their names to make false claims or communications. These maneuvers require the collection of online information about the victim using social engineering techniques and digital espionage. This information is usually collected utilizing scans on social networks, but can also be performed using software such as social engineering toolkit (SET) on Kali Linux (KOYUN; AL JANABI, 2017). Whaling phishing is a type of spear-phishing, but it more specifically targets individuals in high positions in companies. Finally, corporate email compromise phishing is similar to whaling phishing with the difference that its ultimate purpose is to gain access to the mailbox and other private information. In this type of attack, social engineering techniques are combined with the spread of malware by sending attachments containing trojans, keyloggers, and other malware.

### 2.3 MALWARE

According to the glossary of terms found in National Institute of Standards and Technology (NIST) (KISSEL, 2011), malware can be classified as a program which is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system. In its most recent version, the glossary of terms states that malware can be all hardware, firmware, or software which is intentionally included or inserted in a system for a harmful purpose. Furthermore, malware can be seen as a type of program just as any other, and it is necessary to know the types of actions performed by it in order to detect an infection process (PRADO et al., 2016). In addition, it is worth noting that most malware infections are triggered by social engineering, as described by Krombholz, Hobel, Huber, Weippl (KROMBHOLZ et al., 2015).

In the wake of the evolution of malware, a topic which has been drawing significantly more attention in cybersecurity is ransomware, which is directly linked to phishing attacks and other social engineering attacks. Ransomware has been one of the most notorious malware targeted at end-users, governments, and business organizations in recent years. Ransomware can be classified as a type of malware designed to facilitate different nefarious activities, such as withholding personal data unless a ransom is paid. The aforementioned is commonly known as information hijacker (BEAMAN et al.,

2021). It has become a very profitable business for cybercriminals capturing millions of dollars in revenue while also creating a serious threat to organizations through financial losses worth billions of dollars. The number of ransomware attacks has grown exponentially due to easily available ransomware toolkits and ransomware as a service (RaaS) which enables novices to launch (SHARMEEN et al., 2020) ransomware attacks. Since ransomware is already prevalent on personal computer's (PC) and laptops, while also becoming more prevalent on mobile devices, it is already reaching internet of things (IoT) / cyber-physical systems (CPS) scenarios (BEAMAN et al., 2021).

Ransomware can be classified into three main categories: Locker, Crypto, and Scareware (ANDRONIO et al., 2015) (Figure 4). Scareware can generate pop-up ads to manipulate users into assuming they are forced to download certain software, thereby using coercion techniques to download malware. In scareware, cybercriminals exploit fear instead of locking the device or encrypting any data and not damaging the computer system. A ransomware locker has its main foundation blocking the computer's primary functions, thus being able to encrypt certain files which can block the computer screen and/or keyboard. By and large, the maneuvers performed by the locker can be resolved more easily by restarting the computer in safe mode or running a virus scanner. The crypto-ransomware encrypts the user's confidential files but does not interfere with basic computer functions. Unlike cabinet ransomware, crypto-ransomware is often irreversible as it uses current encryption techniques (Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA)) and is nearly impossible to reverse.

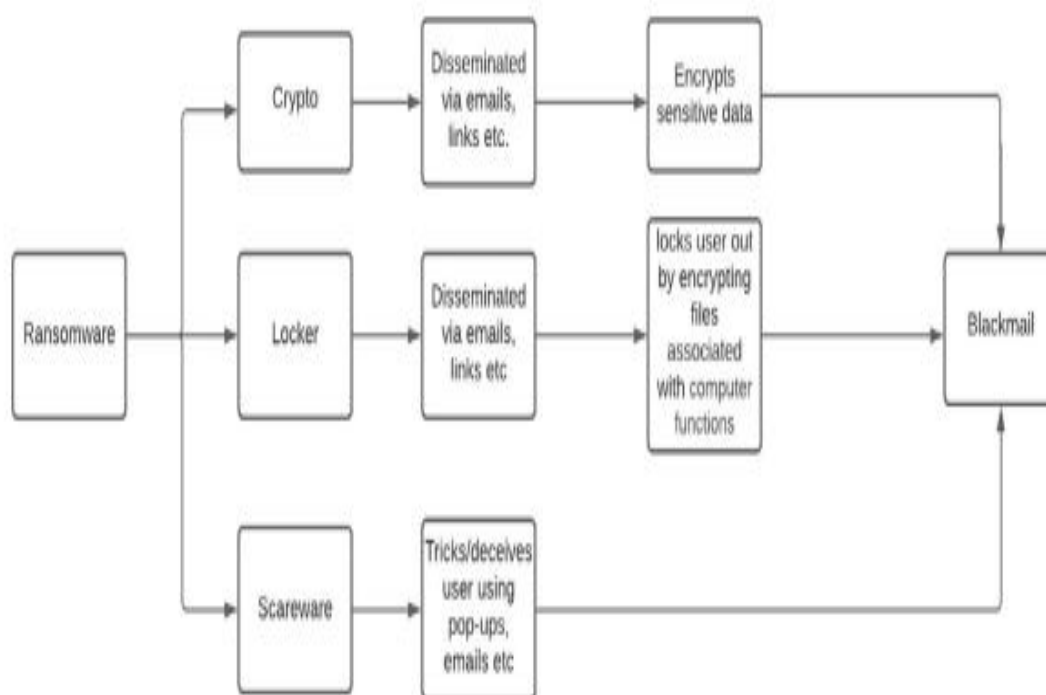


Figure 4 – Types of ransomware (ANDRONIO et al., 2015)

Ransomware can utilize one of three encryption schemes: Symmetric, Asymmetric, or Hybrid (CICALA; BERTINO, 2020). The symmetric approach is problematic as a cryptographic key must be incorporated into ransomware, making this approach vulnerable to reverse engineering. Asymmetric encryption is slow compared to symmetric encryption. As such, encrypting larger files can become a major issue. The most effective approach is hybrid cryptography, which uses both symmetric and asymmetric cryptography. In hybrid cryptography, the first step is to create a random symmetric key by calling a cryptographic application programming interface (API) on the user's operating system. The symmetric key encrypts the victim's files as the ransomware traverses the file system.

After all files are encrypted, a public-private key pair is generated by a command and control (C2) server to which the ransomware connects. The public key is sent to the ransomware and is used to encrypt the symmetric key, while the C2 server keeps the private key. The plain text version of the symmetric key is then deleted to ensure the victim cannot use it to recover their files. Instructions on how to pay the ransom are left for the victim. If the ransom is paid, the decryption process will begin. Decryption commences by requesting the private key from the C2 server. Once obtained, the private key is used to decrypt the symmetric key. Finally, the symmetric key is used to retrieve the victim's files (BEAMAN et al., 2021). Figure 5 describes the step-by-step sequence performed by a hybrid ransomware, from infection to encryption of information.

## 2.4 TAXONOMY IN CYBERSECURITY

Bearing in mind the evolutionary process of threats, alongside the wide range of possibilities for exploiting these threats, the approach of integrating measures with information security to detect and map computer threats is essential. The use of a taxonomy in the classification of threats and vulnerabilities can serve to choose the best countermeasures. This approach took place in the early 1990s when the importance of using taxonomy in cybersecurity was highlighted through the work of Lindqvist, Jonsson, and Bishop (BISHOP et al., 1995), (LINDQVIST; JONSSON, 1997). It was ascertained that categorizing and classifying threats could be a solution, expanding knowledge about the phenomenon and making isolated studies possible (GUPTA et al., 2018), (NARWAL et al., 2019). As presented in (HOWARD; LONGSTAFF, 1998), (KRSUL et al., 1998) the process to develop a good taxonomic structure must satisfy some characteristics such as objectivity, determinism, and acceptable and useful reproduction. The various taxonomic schemes presented in the works analyzed can gather information about vulnerabilities and threats. Existing classifications are important to determine whether an object analyzed represents something new or already known along with its exploration methods.

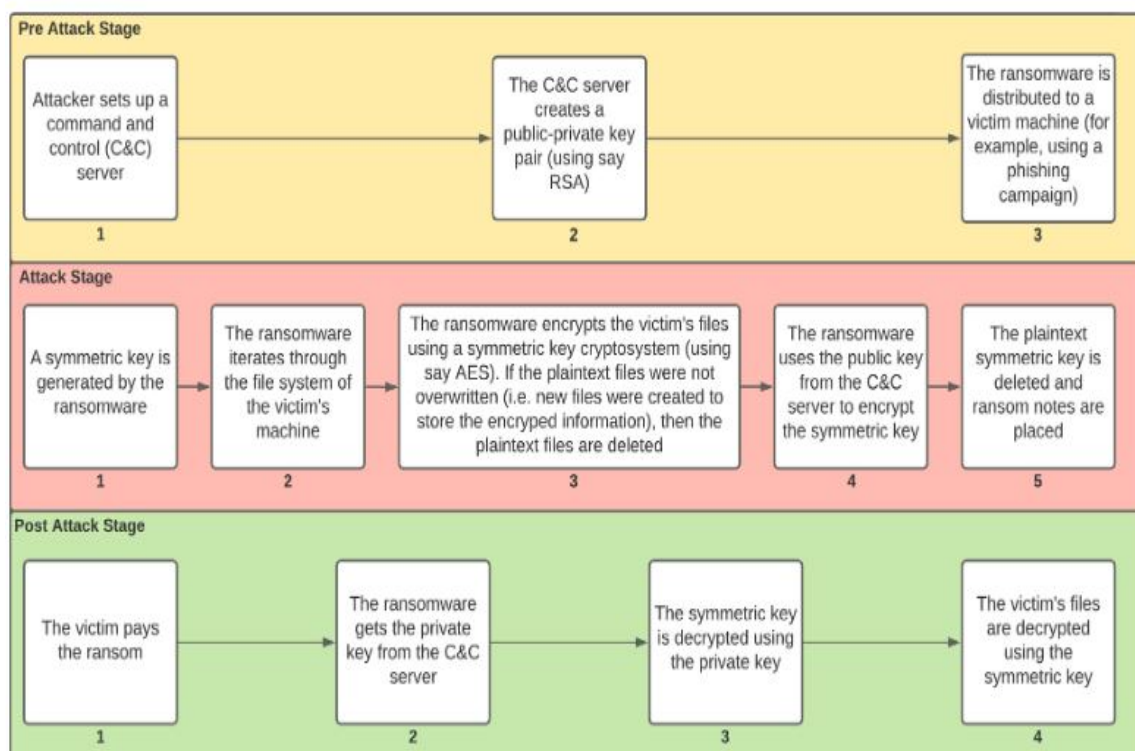


Figure 5 – The typical steps used by ransomware to encrypt and decrypt a user's data. (BEAMANet al., 2021)

As a means to deal with the various threats, there are several robust taxonomies in the literature. Each taxonomy has a set of advantages, and consequently, its limitations (SADQI; MALEH, 2021). In the work of Sadqi and Maleh 2021, an analysis of several taxonomic structures found in the literature was performed, focusing primarily on the scenario of web applications. Through each analysis, it is possible to determine the advantages and disadvantages of each taxonomic structure. The taxonomy proposed in the study has taken advantage of the benefits of existing taxonomies and provided an integrated approach to classifying client-side and server-side attacks (figure 6). According to the authors, a web security threat or issue is defined as a potentially malicious activity which uniquely targets one or more components of the web application's architecture, such as the user's browser or the web application hosting server. The security community agrees that rating security issues and drafting threat taxonomy can help educate software developers and security experts to better understand the root causes of software failures and build more secure web applications.

The usage of classification structures in combating cyber threats represents a useful albeit insufficient tool for practical detection and prevention measures. Detecting patterns of maneuvers, such as phishing through intelligent systems, has become prominent within computational security given the advance in the complexity of threats which represent an arduous challenge to be faced (ABUZURAIQ et al., 2020). Taking into consideration the threats from social engineering (Figure 7), this challenge

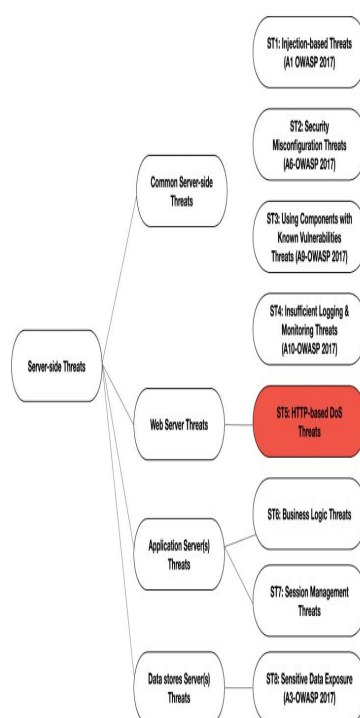


Figure 6 – Server-side web threats (SADQI; MALEH, 2021)

becomes even greater, as described in the work of Aldawood, Hussain, and Geoffrey Skinner (ALDAWOOD; SKINNER, 2020). The investigations around phishing need to be highlighted as they are considered one of the main criminal maneuvers practiced on the Internet. However, the best ongoing form of protection is to suspect any email or page with unknown links and files. Thus, phishing is an example of how threats, which act directly through social engineering, are considerably more difficult to mitigate.

## 2.5 AUTOMATA THEORY - DETERMINISTIC FINITE AUTOMATON (DFA)

Automata theory can be viewed as one of the oldest and most researched areas of Computer Science (KHOUSSAINOV; NERODE, 2012). The aforementioned study points out that in the last 50 years countless automata applications have been developed in a wide spectrum of areas with a corresponding evolution from a variety of theoretical models. The first applications of automata theory included pattern matching, syntax analysis and software checking, where elegant theory was applied to real-world problems, resulting in the generation of useful software tools mainly in the area of compilers. Deep connections between automata and the logic continue to be discovered, and newer models of automata, such as timed automata, hybrid automata, distributed automata and weighted automata were all proposed and driven by specific applications.

In accordance with Michael Sipser in (SIPSER, 1996), the theory of automata addresses the definitions and properties of mathematical models of computation, thereby



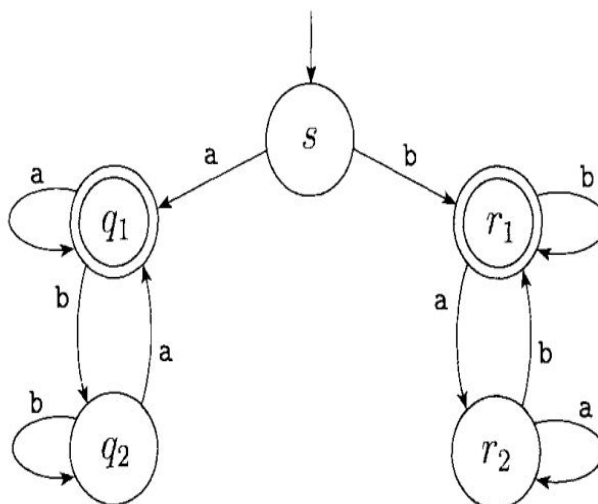
Figure 7 – Taxonomy of social engineering attacks and threats (ALDAWOOD;SKINNER, 2020)

being able to use these in several areas of the field. Furthermore, the theory of automata allows important concepts to be grounded in other non-theoretical areas of computational science. Finite automata (DFA), or finite state machines, are used in computers with a very limited amount of memory and are highly useful tools for recognizing patterns in data. Observing the theoretical construction on automata proposed by Sipser, it is worth noting that finite automata can be represented in an abstract way (not a precise representation) through state diagrams. However, they also have a formal mathematical definition (representation resolves to a finite automaton).

The formal definition states that a finite automaton consists of a list of five objects (5-tuple): set of states ( $Q$ ), input alphabet  $\Sigma$ , rules for movement  $\delta$ , initial state ( $q_0$ ), and acceptance states ( $F$ ). It is worth noting that automata can be used to monitor computational actions, such as monitoring the behavior of a specific object. Actions can be represented through states reaching a state of acceptance, this one representing the reading compatible with its purpose. DFA's can have the initial state as the accept state. Therefore they will accept the empty entry  $\epsilon$  in their alphabet. Figure 8 demonstrates a DFA with the alphabet  $\Sigma = (a,b)$  and two accept states,  $q_1$  and  $r_1$ . This DFA begins



in state  $s$  ( $q_0$ ), and after reading the first input symbol, it will either go left ( $q$  states) or right ( $r$  states). The  $M_4$  automaton will accept all strings that start or end with 'a' or start with 'b' and end with 'b' (Strings that start and end with the same symbol).



**FIGURE 1.12**  
Finite automaton  $M_4$

Figure 8 – Exemple of Finite Automaton  $M_4$  -(SIPSER, 1996)

## 2.6 WEB CRAWLING

As set forth in (OVELGÖNNE et al., 2017), Web crawling is the process used by search engines to collect web pages. This investigation looked at web crawling at several different levels, from the long-term goal of crawling important pages first, to the short-term goal of using network connectivity efficiently, including implementation issues which are essential for tracking in practice. There is also an algorithm used by search engines to find, read, and index pages on a website, which is known as a web spider. It is like a robot which captures information from each link that it finds, registers, and understands information according to specified criteria in search engines. This also facilitates the analysis of the code of a website to look for information on a page.

According to (OLSTON; NAJORK, 2010), the main function of a crawler is to examine links across the internet. Simply put, it does a complete scan of the links that you find on the web, taking into account each line of code on the site and all the links that are on it, whether internal or external. Thus, it can build a sort of internet map with the right to all the sites which link to each other. Among the other functions of a crawler, the following are notable: evaluation of sites, copying the structure of sites for

indexing, identifying new sites, and carrying out automated maintenance tasks such as link checking and HTML code validation. It all commences with a list of URLs (seeds) to visit the online web crawler. On every visit to these sites, the robot identifies the links on the pages and includes them in specific lists for a new scan. Then it returns to them recursively, as per the established rules. And if it finds new content, it gets indexed. Your search engine ranking may change if you find updated content on a preexisting page. Figure 9 provides an example of a web crawler architecture.

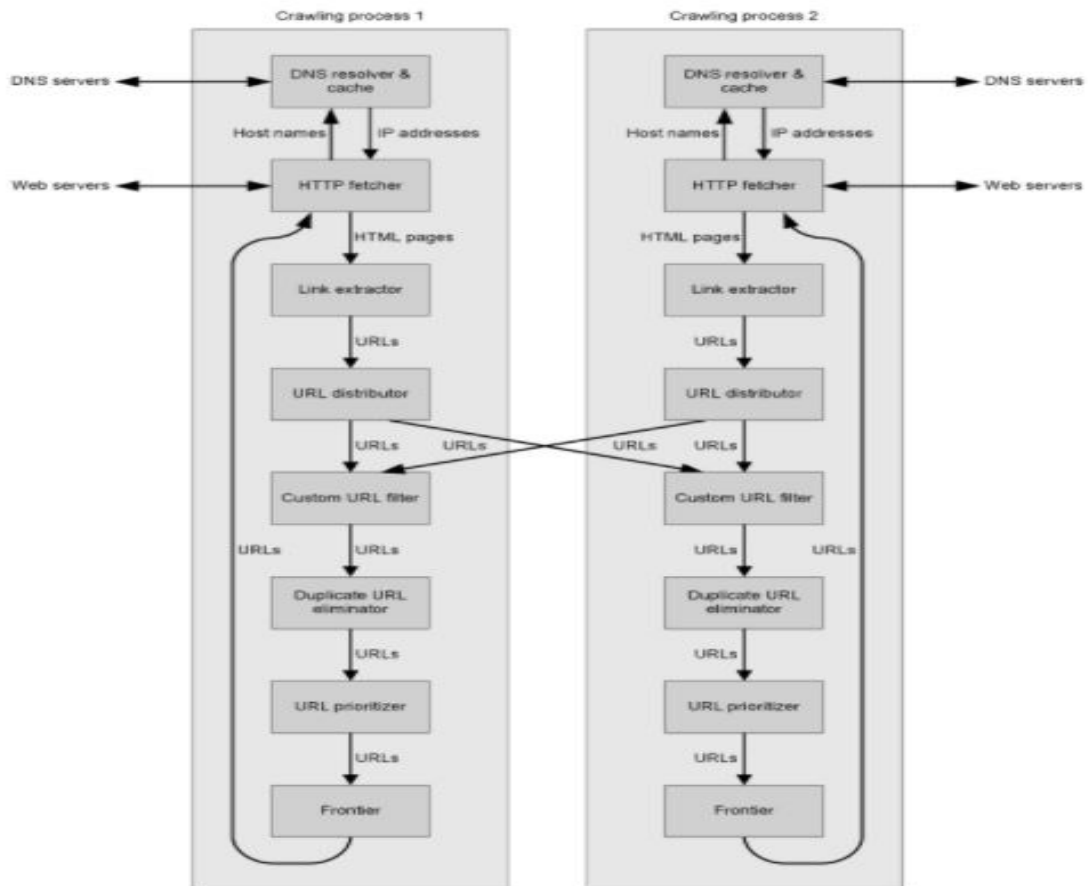


Fig. 2.1 Basic crawler architecture.

Figure 9 – Example of Crawler Architecture - (OLSTON; NAJORK,2010)

### 3 RELATED WORKS

This chapter contains the bibliographical review pertaining to the state of the art of the themes with respect to this work. Section 3.1 presents a systematic review. Section 3.2 lays out the research works related to the development of phishing detection techniques and methodologies. Lastly, Section 3.3 addresses the work on mapping user computational behavior in cybersecurity.

#### 3.1 SELECTED WORKS AND SYSTEMATIC REVIEW

For the purpose of surveying state of the art regarding the themes investigated in this work, specific keyword searches were conducted, making it possible to observe a wide range of works related to the themes presented here. These works were sought directly on the Google Scholar database, with no restriction on publications in other databases.

First of all, a search was conducted on the subject of Phishing Detection. The base search string used was only in articles within the period of 2016-2020. This string was configured to perform a search for words specifically in the title of the articles. Approximately thirty-seven articles were returned from this search string. Of these, nine were chosen which presented approaches relevant to the focus of this study. These nine works were considered as the State of the Art.

The inclusion criteria for the articles were: a publication period between 2016 and 2021, full articles published without a language restriction, and articles which proposed the computational implementation of some real-time phishing detection mechanism. Thereafter, the selected works on phishing detection were commented on. At the end of the session, a comparative table between the research works and this study is presented.

Lastly, a search for works related to the second stage of this study was conducted, whose main focus is investigating ways of mapping the user's computational behavior and its risks related to cybersecurity. As a means to do this, a search was first performed using the string ("Phishing Detection") AND ("User Behavior") in the titles of articles or any location of the text.

The string was created to unite the two themes, but it did not return satisfactory results. Afterward, the string ("User Behavior") AND ("Security") was used in the period between 2016-2021. The string obtained a return of approximately 49 results. Of these articles, the most relevant ones were evaluated in terms of citations and with greater compatibility with the theme, resulting in eight articles. Following the same logic used previously, the articles were discussed and arranged in a comparative table.

## 3.2 PHISHING DETECTION

In this section, studies related to phishing detection methodologies will be discussed, which is the main topic of the first stage of this study, thereby addressing the main aspects and differences regarding this investigation. A comparative table of research works related to this study will be presented (Figure 10).

Threats, such as Phishing, have attacked vectors and specific features which enable their detection. By mapping these vectors, the key points can be detected which need to be checked in URLs and e-mail messages in order to recognize phishing maneuvers. These vectors represent patterns already known in malicious URLs which can be recognized by collecting world wide web (WEB) data. The scraping technique allows data extraction from pages in search of specific characteristics and can assist in validating URLs' legitimacy and in detecting phishing vectors (PARK et al., 2017). Similarly, searches for specific strings can be performed in e-mail messages to obtain the same patterns checked in the URLs.

Anti-phishing solutions provide additional indicators that a website may be fraudulent. There are a variety of clues to avoid falling into phishing maneuvers along with malicious URL detection solutions (ALMEIDA; WESTPHALL, 2020). As demonstrated in (SILVA et al., 2020), phishing detection techniques can be classified into two groups based on the following: lists or heuristic techniques. List-based techniques can use a resource base appertaining to permissive listings (whitelists) or restrictive listings (blacklists). Heuristic techniques extract the characteristic properties of existing phishing pages to detect new fraudulent pages. The main advantage of heuristic techniques is that they can detect newly created or unknown (zero-day) phishing attacks, and list-based techniques cannot (RAO; PAIS, 2019).

The work done by (SILVA et al., 2020) focused primarily on predicting phishing attacks based on a set of static resources and observing the occurrence of these resources in current phishing maneuvers. The analysis described static elements such as keywords and patterns in phishing URLs. The methodology applied put forward a set of twelve characteristics submitted to three distinct samples of legitimate and phishing sites during the year 2018. The evaluated characteristics could be divided into three groups: URL blacklist bypass, URL morphology, and user susceptibility.

It is worth noting that some of the characteristics described by the authors will be used as a basis for the search engines proposed in this work, namely: concatenate subdomains, homographic attack, and URL with redirection. The study revealed that even though research on phishing detection and prediction has developed considerably, certain characteristics are of low relevance, and others have not kept up with the changes in the computational scenario. The authors pointed out that some features are found more regularly in phishing and could be exploited more efficiently, indicating that further investigations need to be conducted. In addition, the study also undertook

a qualitative analysis of behavior, managing to identify aspects such as relevance, relationships, and similarities between resources.

In (RAO; PAIS, 2019), the study proposed Jail-Phish, which uses mechanisms capable of detecting phishing URLs hosted on compromised servers (PSHCS) and legitimate websites recently registered. The structure compares the suspect page and the corresponding domain in the search results to calculate the similarity score between them, highlighting the difference between PSHCS pages and legitimate pages as high. The work utilized search engines in URLs in real-time, as in this study, but it did not present effective mechanisms to mitigate attacks. Another point of divergence was that this research did not focus on the detection of PSHCS; our work proposes ways to detect phishing vectors regardless of where the pages are hosted. The work did, however, show an accuracy of 98.61 % and made use of a hybrid methodology, mixing lists and heuristic techniques.

According to (JAIN; GUPTA, 2018), the most common way to run phishing attacks is to send thousands of fake emails to users, prioritizing a large number of short URLs online. The study proposed a phishing search engine which performed two authentications at different levels before declaring a page as fake. Queries performed searches for the title along with the page's domain. The authentication processes would then observe values within the source code of the page as hyperlinks to assist in detecting phishing characteristics, thereby increasing the accuracy of the method. The work exhibited good accuracy in the detection, but it did not present a mechanism to mitigate malicious actions.

The investigation done by (PARK et al., 2017) put forward Phishing-Detective; a heuristic structure for detecting phishing based on a Web Crawler which extracts information from pages using scraping techniques. The extracted data is analyzed by a data mining tool (RapidMiner) to find patterns and report findings. The proposed structure made use of the Scraping technique, similar to this study. However, it used a base (PhishMonger) which is outdated for reproducing current methods, in addition to not showing significant results in the detection of URLs or the approximate time to perform the checks.

In (CHIEW et al., 2018), a description is given related to an extension from a previous study in 2015, where logo images were used to determine the consistency of the URLs' identity; the actual and the portrayed. Meters were used to determine a consistent identity as a legitimate page and an inconsistent one as a phishing page. Part of the detection carried out was done through machine learning, based on the extracted logo image. Finally, a comparison between the domain name returned by Google and the name of the consulted URL enabled us to differentiate phishing from a legitimate page.

In (SAHINGOZ et al., 2019), a real-time anti-phishing system was proposed,

which applied different classification algorithms and resources based on natural language processing (NLP). The system had remarkable properties such as language independence, real-time execution, and independence from third-party services. The Random Forest algorithm was used in the elaboration of classifiers and resources, reaching an accuracy of 97.98 % in detecting phishing URLs.

According to (SOUZA et al., 2019), the authors advocated PhishKiller, a tool for detecting and mitigating phishing pages. The tool receives the addresses accessed by users through a redirection made by the proxy and uses concepts of deep learning (DL) to classify the URLs received. According to the results presented, the methods achieved an accuracy of 98.3 % in the detection of malicious addresses.

In (DOBOLYI; ABBASI, 2016), the authors commended PhishMonger, a URL verification API, in sync with the PhishTank platform. The API is responsible for synchronizing with Phishtank and downloading files from the suspicious page to the last depth level. This approach can present a high computational cost in more complex sites or significant variation in the file format. Contrary to what is proposed by the authors in this study, specific searches for Phishing elements would be carried out through Web Crawling, thus avoiding the reading of trivial data in Phishing detection.

WORK	LISTS	PLATAFORM	METHODOLOGY	ACCURACY	LANGUAGE INDEPENDENCE	PHISHING IN COMPROMISED SERVERS
Park et al. (2017)	Blacklist	PhishTank PhishMonger	String Search Engine, Scraping and Datamining	-	Yes	No
Jain and Gupta (2017)	Blacklist	PhishTank OpenPshish	Dynamic Strings Search Engines	99,95%	Yes	No
Chiew et al. (2018)	Whitelist Blacklist	PhishTank	Machine Learning and String Search Engines	96,93%	Yes	Yes
Rao and Pais 2019	Whitelist Blacklist	PhishTank	Dynamic Strings Search Engines	98,61%	Yes	Yes
Sahingoz et al. (2019)	Whitelist Blacklist	PhishTank Yandex	Machine Learning and Natural Language	97,98%	No	No
Souza et al. (2019)	Whitelist Blacklist	PhishTank, OpenPhish, Phishbank and Malware Domain List	Deep Learning	98,3%	Yes	No
Santos et al. (2019)	Honeypots	-	Naive Bayers + Lexical Analysis of URLs	-	No	No
Dobolyi et al. (2016)	Blacklist	PhishTank	String Search Engine, Scraping and Web Crawling	-	Yes	No
Silva et al. (2020)	Valid Phishing Pages Invalid Phishing Pages	PhishTank	Scraping and Web Crawling	-	Yes	No
This Work	Whitelist	PhishTank	String Search Engine, Scraping and Web Crawling	97,66%	Yes	No

Figure 10 – Phishing Detection - Comparative Table of Works Related to this Work

Finally, in (SANTOS et al., 2019), malicious emails were collected through active honeypots. For the detection of phishing messages and differentiation of spam messages, the Naive Bayes classifier was used, with a 92 % hit rate. To extract phishing URLs, heuristic techniques for searching for specific characters in their bodies were used. If any of the characters sought were found, the URL would be kept in the analysis group.

### 3.3 USER BEHAVIOR

The examination of user computational behaviour and the human factor is not only present in information security, based on the work of Cialdini (CIALDINI; JAMES, Lloyd, 2009), but the topic is widely studied in the areas of social psychology and behavioral analysis, addressing topics concerning how susceptible the user is to phishing techniques and other manipulation maneuvers (SHENG et al., 2010),(MOHEBZADA et al., 2012). On the other hand, several studies show that characterizing the behavior of individuals to obtain the level of susceptibility of these individuals to generate vulnerabilities is not feasible, as quantifying these factors and adapting them in a computational way is not a simple task.

As described by NIST 2018, in its cybersecurity education framework (BARRETT, 2018), user awareness remains the greatest way to combat social engineering. Authors Zwilling, Moti, et al. and Ramlo, Susan E., John B., and Nicholas demonstrate that increasing security maturity is the first step in preventing Internet-related risks (ZWILLING et al., 2020), (RAMLO; NICHOLAS, 2020). The user needs to be aware of these risks, adopting a preventive posture and attention to safety, thereby incorporating habits into the routine of all users.

No matter how robust the security solutions are, they render useless if not accompanied by a minimum level of awareness. Increasing the user's maturity level in information security results in an exponential drop in vulnerabilities generated by the user in a computational scenario. Nevertheless, within this category, other studies also point to the intention factor as another variant, and it is directly linked to behavior (ADDAE et al., 2019), or even the violation of rules determined in an organization's security policy (THAKUR; PATHAN, 2020).

The topics investigated in (OVELGÖNNE et al., 2017) demonstrate that all evaluated (binary) datasets are related to the number of pieces of malware found on host machines without a statistically significant level. Of these statistically significant results, the ones which were considered more solid indicate that the number of malware infections on a machine is related to the number of binaries downloaded. Also, the number of binaries on hosts is linked to the number of malware infections on hosts in the software developer category, and software developers can generate binaries by compiling the code they are developing. Software developers can generate binaries by compiling the

code they are developing. In addition, five-user groups were identified (players, software developers, professionals, others who are not in any of the above categories, and all users), and it became evident that software developers seem to be the most prone to malware attacks.

The study (ISKHAKOV et al., 2018) set forth an approach where modern automated systems clarify the imperfection of existing approaches in developing technologies for identifying user behavior and malicious maneuvers. It was evident that the use of thermal maps does not ensure accurate and authentic identification of a particular user, but it does possibly raise the likelihood of detection in combination with other identification methods which allow, with a certain probability, the comparison of users applying various means of anonymization.

In (MARTIN et al., 2018) the main objective of the work was to examine the usefulness of the equal equal variance sign detection theory (EVSDT) for assessment along with understanding human detection of phishing and spear-phishing email scams. Through an online inbox simulation, this work's results suggested that differences in susceptibility to phishing and spear-phishing emails could be carefully quantified for detection accuracy and response bias through the use of an EVSDT framework. Furthermore, the results indicated that based on EVSDT, point metrics are effective for modeling and measuring susceptibility to phishing in the inbox task, without the need for parameter estimation or model comparison involving unequal variance SDT (UVSDT).

The probe conducted in (LÓPEZ et al., 2019) put forth a diagnosis of the cybersecurity situation in Spanish digital homes. This was done by analyzing the adoption of security measures and the level of incidence of situations of which may pose security risks. In this study, the trust of home users is also evaluated through computer analysis, which determines the degree of malware infection. The tool selected to perform the analysis was the self-organization map (SOM), a data mining method which provides a low-dimensional representation of high-dimensional data, thus improving visualization and interpretability for complex pattern recognition. Furthermore, the study presented two types of the applied methodology.

Firstly, a general analysis of a specific questionnaire on cybersecurity was carried out. Secondly, automated scanning software was used. The purpose of using the questionnaire and data independent of the scanning software was to reduce possible bias in user responses regarding perceived security. This problem, common to all types of surveys, is known as optimistic bias and is especially important for user-related issue concerns. After the data was collected from both sources, it was then pre-processed by grouping it into categories and extracting resources of interest. Finally, SOMs were applied to extract knowledge about the interactions between these resources.

In (ADDAE et al., 2019), the academic analysis described an exploratory investigation on the feasibility of predictive methods. User behavioral data was analyzed as a



possible aid in developing effective user models for adaptive cybersecurity. Partial least squares structural equation modeling was applied to the cybersecurity domain, collecting data on users' attitudes towards digital security and analysis of how this influences the adoption and use of technology security controls.

Bayesian network modeling was then applied to integrate the behavioral variables with simulated sensory data and/or records from a web browsing session and other empirical data collected to support personalized adaptive cybersecurity decision making. The results of the empirical study show that predictive analytics is feasible in the context of behavioral cybersecurity and can help generate useful heuristics for the design and development of adaptive cybersecurity mechanisms.

An observational study was conducted by (DIAZ et al., 2020) on the relationship between demographic factors and phishing susceptibility in the university of Maryland (UMBC). Four hundred and fifty randomly performed phishing attacks selected students on three different days (1,350 students in total) to examine user click-through rates and demographics across UMBC's undergraduate students. No significant correlation was found between gender and susceptibility. In addition, it was noted that there is a reverse correlation between phishing perception and the student click resistance.

Students who identified themselves as understanding the definition of phishing had a higher susceptibility rate than their peers who were only aware of phishing attacks, with both groups having a higher susceptibility rate than those without any knowledge at all. Approximately 70% of respondents who opened a phishing email clicked on it, with 60% of students having clicked in general.

In (ALJEAID et al., 2020) the study focused on assessing the level of cybersecurity knowledge and cyber awareness in Saudi Arabia. It was aimed at assessing end-user susceptibility through three phishing attack simulations. In addition, the study elaborated on some of the concepts related to phishing attacks and reviewed the steps required to initiate such attacks.

In (MASHIANE; KRITZINGER, 2020), theories from the domain of psychology were incorporated to understand cybersecurity user behavior. Firstly, the study put forth the question as to which of these theories is best suited to understand cybersecurity behavior, and subsequently, to change cybersecurity behavior for the better. The research established a definition for the different categories of cybersecurity behavior to answer this question. In addition, it identified and applied a structure, the theoretical domain framework (TDF), and united different behavioral theories in a behavior change framework.

TDF was used as a tool to synthesize the constructions found in the literature. For each category of behavior (identified in this study) the associated constructs were identified and evaluated to determine whether they are facilitators or barriers to behavior. Through this, the study aimed to show the link between the behavioral constructions

discussed in Theoretical Domain Structure for different categories of cybersecurity behavior. As a final result, the contribution of the study served to implement initiatives aimed at changing cybersecurity behavior.

WORK	PROPOSAL	DATASET	METHODOLOGY	PERIOD
Ovelgonnie et al. (2017)	identifying human behaviors which increase the risk of malware attacks on a host based on real-world operational data	Datasets extracted from WINE Symantec (binary reputation and antivirus telemetry)	Subdivision of users into profiles with different scores and subdivision of behavior into analysis categories such as number of attacks per host and most attacked hosts	January - August 2011
Iskhakov et al. (2018)	Observe the security incident rates in applications that do not require authentication from the main part of the users	Thermal Points collected in web-browsers	Use of thermal maps adapted to user profiles and the "user-mouse" system	2017
Martin et al. (2018)	Examine the utility of equal-variance signal detection theory (EVSDT) for evaluating human detection of phishing and spear-phishing e-mail scams.	E-mail response of 344 Individuals	Quantify susceptibility to phishing and spear-phishing emails through an EVSDT (Equal-Variance Signal framework Detection Theory)	2017
Lopez et al. (2019)	Understand the user's computational behavior related to security incidents through the application of questionnaires divided into specific categories	9181 surveys and 6350 computer scans in total	Self-Organizing Maps (SOMs)	December 2013 - June 2014
Adae et al. (2019)	Investigate the feasibility of predictive methods analysis of user behavioral data as a possible aid in developing effective user models for adaptive cybersecurity.	421 survey responses of a mix of 50 university students	Partial least squares structural equation modeling with Bayesian-network	2018
Diaz et al. (2019)	Conduct an observational study study on the relationship between demographic factors and phishing susceptibility in the University of Maryland, Baltimore County (UMBC)	Click-through rates and demographics of 1,350 undergraduates in response to 450 phishing emails sent	Observe the users' behavior towards three categories: the billing problem, the contest winner and expiration date phishing tactics	Spring 2018
Aljeaid et al. (2020)	Assess the level of cyber security and cyber knowledge and awareness in Saudi Arabia end-user susceptibility through three phishing attack simulations.	66 participants in a targeted phishing experiment	Classify phishing attacks and their steps, performing a comprehensive analysis to assess user behavior.	2019
Mashiane and Kritzinger (2020)	Show the link between the behavioral constructions discussed in Theoretical Domain structure for different categories of cybersecurity behavior aiming to change computational behaviors regarding cybersecurity	Only Literature review	application of Theoretical Domains Framework (TDF) to analyze changing user behavior	2019
This Work	Map user computational behavior when interacting with classified attack vectors for threats such as phishing and malicious code infection	Computational interactions of 30 users captured in a corporate scenario	Capture the user's computational behavior through a finite deterministic automaton (DFA) capable of generating a cybersecurity maturity tree	2020 - 2021

Figure 11 – User Behavior - Comparative Table of Works Related to this Work

## 4 PHISHING DETECTION AND USER BEHAVIOUR MAPPING

This chapter will demonstrate the implementation in an experimental setting of the method proposed in this research study. This study can be divided into two steps. In the first step (Section 4.1), a phishing detection method (ALMEIDA; WESTPHALL, 2020) will be presented. The method primarily scans URLs and email messages for standard attack vectors in phishing scams and malicious code infection attempts. Thereafter, in the second stage (Section 4.2), monitoring user behavior will be proposed. This method is based on constructing a DFA. It uses the method presented in the first section of this chapter through the implementation of monitoring sensors which collect information about user actions performed in the environment. These sensors will be strategically placed in the same experimental scenario used in Section 4.1, where user actions can be classified and mapped precisely when interacting with threats, such as phishing and malware infection.

Section 4.1. is organized into 4 subsections. Subsection 1 provides a description of the experimental scenario used. Subsection 2 demonstrates the functionality of the implemented collections. Subsection 3 puts forward the proposed metric for evaluating phishing URLs along with the elaborate algorithm. Finally, subsection 4 exemplifies the behavior of APIs and their respective interactions. Section 4.2 is further broken down into four subsections. The first subsection presents a taxonomy of user actions and the classification of attack vectors. The second subsection introduces the new additions to the experimental scenario proposed in section 4.1. The third subsection proposes the finite state machine (DFA) implementation used to monitor user behavior. Lastly, subsection four exemplifies the elaboration of the user maturity tree through the proposed APIs.

### 4.1 HEURISTIC PHISHING DETECTION

The methodology presented in this section intends to demonstrate the detection of phishing pages in real time. That being said, string search engines have applied that aspect to characteristics such as the extraction API proposed in this work. The API is developed based on concepts related to state of the art and incorporates the principle of the scraping technique, which serves as a Web Crawler as it acts as an algorithm which analyzes the code of websites. The algorithm behind the application is also able to scan text files for strings which can indicate URLs, domains, or attached files. The extracted elements will be stored in collections. These represent structures which will contain important information about the results obtained by the search engines.

Another significant contribution of this methodology consists of the fact that it is even using a Dataset imported from another database. The API is still able to perform new tests on the URLs already obtained and also on the new ones, reducing false

positives to the maximum. This type of action allows the detection of zero-day phishing URLs, as each URL captured at the moment is accessed by the user and a comparison is made to find out if it has already gone through the Web Crawler check. Through this methodology, it is possible to analyze which malicious URLs are going unnoticed by the reporting platforms, which types of phishing are occurring the most for a given group, and which themes are widely being used in the composition of the fake pages.

#### 4.1.1 Implementation and experimental scenario

In order to conduct the experiments, free software tools (Anti Spam Zimbra and Firewall PfSense) were used with a Not Only SQL (NoSQL) database (MongoDB). The three tools were configured on separate virtual machines on a Ubuntu operating system. The machines were implemented in a laboratory environment, with 200 users authenticated via Lightweight Directory Access Protocol (LDAP). Users' WEB traffic was directed to the proxy, and their email boxes monitored by Anti-Spam, as shown in Figure 12.

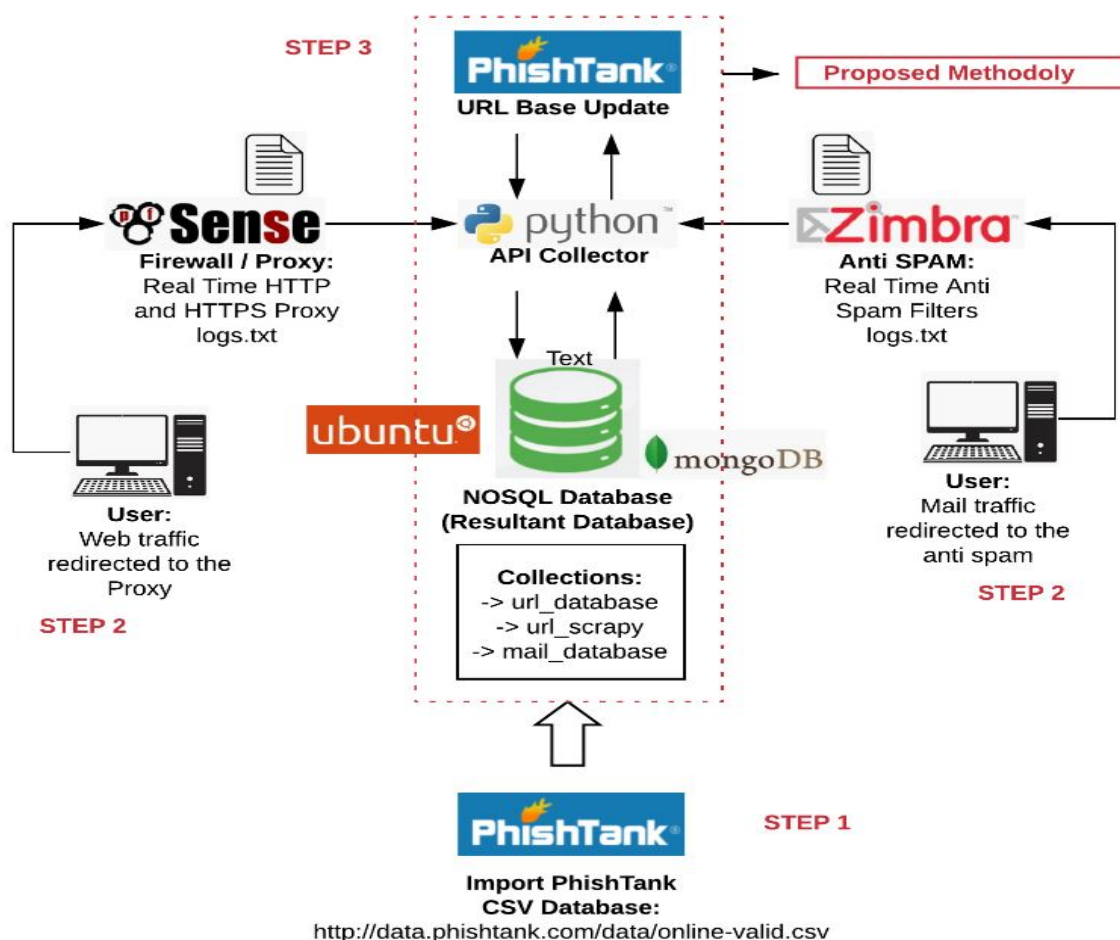


Figure 12 – Experimental Scenario

To carry out the experiments in this work, we used the creation of access profiles

in Proxy based on a Whitelist. Instead of maintaining a base of blocked URLs (blacklist), a list of official URLs has been inserted to prevent occurrences of false positives. The API collects and checks the phishing vectors on an official URL. Correspondingly, the Firewall is programmed to generate an access log only when the user accesses a URL outside the Whitelist.

It is worth mentioning that the choice of this scenario was aimed at maintaining the maximum approximation with the structural reality commonly found in corporations with essential security criteria. Figure 13 shows basic descriptions of the configuration of the virtual machines to reproduce the scenario used.

VIRTUAL MACHINE	OPERATING SYSTEM	VCPUs	MEMORY (GB)	DISK SPACE (GB)	IP
 API Collector	Ubuntu 20.04 LTS	8	16	2048 SSD	172.16.30.10/24
<p><b>Considerations:</b> Use of Python2 to implement the Collector API and its modules (url_scrapy, url_database, mail_database) - HMAC library in Python - Using Mongo DB for Centralized Database with three initial collections (url_scrapy, url_database, mail_database) - Import from PhishTank Database in csv format and insert into url_database - Implementation of Syslog for reading logs (Firewall) and (AntiSpam) - If this machine is used with a log centralizer (syslog - Recommended) may need more storage capacity (GB) - API can be configured to read logs directly from machines (inform log IP/Directory - Not recommended, requires more processing power and network throughput) - DNAT rule in the firewall machine to release Collector API ports for WEB and communicate with PhishTank</p>					
	Pf-CE-2.4.x or above	8	8	1024	LAN - 172.16.30.254/24, WAN - 192.168.10.254/24
<p><b>Considerations:</b> Disable Block Private Network if firewall is behind Modem - Check for updates - 10Gbase-T Full Duplex interfaces - Enable Secure Shell (Port 22) - Enable Console Menu - Disable all IPV6 settings on interfaces to save resources - Enable automatic Configuration backups - The WAN interface must be the same as the machine hosting the VM, if the structure is being used on a server, a DMZ or BRIGE must be done with the MODEM on the port used - Proxy browsing logs can be used in text format on the firewall's virtual machine or exported in syslog format to the machine where the syslog was configured (inform IP and directory to send - RECOMMENDED) - LDAP integration with the domain controller machine for user access control.</p>					
	8.8.15 or above, Spam Assassin 3.4.x or above	8	16	1024	172.16.30.20/24
<p><b>Considerations:</b> The Anti Spam tool can be configured on a separate machine due to resource consumption (Recommended) - Logs can be used in txt format on the virtual machine where zimbra was installed or exported in syslog format to the machine where syslog was installed configured (inform IP and directory to send - RECOMMENDED) - Other Open Source Anti Spam tools Can be used, it is not mandatory to use Spam Assassin - The requirement of the email platform and AntiSpam is to read the email body and header in text format and export the log in txt or syslog format (recommended)</p>					
	Windows Server 2012 R2 or above	8	8	500	172.16.30.30/24
<p><b>Considerations:</b> The machine will be used as a domain controller (Active Directory - testdomain.com) and DNS server - Users must connect to the domain to perform the tests - The test domain must be integrated via LDAP with the firewall</p>					

Figure 13 – Virtual Machines and Configurations

Initially, for the creation and population of the NoSQL database, importing the PhishTank Database in Character-separated values (csv) format was used (Figure 14). The choice of the Phishtank base among other platforms was made due to the availability of access to the platform's records and the possibility of connection through its API to carry out queries to the base.

```

2 from pymongo import MongoClient
3 client = MongoClient('localhost', 27017)
4 db = client.Phishtank
5 url_database = db.url_database
6
7 ## Phishtank requisits ##
8 import requests
9 import base64
10 import json
11 api_url = "https://checkurl.phishtank.com/checkurl/"
12
13 logs = open('firewall_Test.txt', 'r')
14
15 for log in logs:
16     try:
17         index = log.strip().index('URL:')
18         url = log.strip()[4:]
19         registro = url_database.find_one({"url":url.strip()})
20
21         if registro is None :
22             #Cria o payload com a url a ser pesquisada
23             payload = {'app_key': 'afda34b2cf34af3c45e14b0487dc3f4720231e19b0c837384725d6f5760194f5',
24                       'url': base64.b64encode(url.strip()),
25                       'format': 'json'
26                     }
27             try:
28                 # Requisição POST ao Phishtank
29                 response = requests.request("POST", api_url, data = payload)
30                 registro_phishtank = json.loads(response.text.encode('utf8'))
31
32                 #Verifica URL no Phishtank
33                 existe = registro_phishtank['results']['in_database']
34                 if existe:
35
36                     inserted = url_database.insert_one(registro_phishtank['results'])
37                     print inserted
38
39             except:
40                 print "Erro ao conectar a api phishtank "
41         else:
42             print "URL found in Database"
43     except:
44
45         #nao encontrou a string URL na linha /log
46         #bypass
47         continue
48
49 logs.close()

```

```

root@dev:/var/www/blip-teste/phishtank# python api-collector.py
url found in database
url found in database
url found in database
url found in database
<pymongo.results.InsertOneResult object at 0x7fc3d9531518>
new url stored in database

```

id	url	check_url	url	check_data_url	in_database
1	http://www.sengpracticesecurity.com/conferences	False	http://www.sengpracticesecurity.com/conferences	False	False

Figure 14 – Connections between python API and Phishtank Database

#### 4.1.2 Collections

This study proposes the use of three collections, which represent abstractions of the elements analyzed by the search engines: **url-database**, **url-scrapy**, **mail-database**. To extract the elements and feed the collections, an API written in Python is used, whose main functionality is the validation of the URL checking methods proposed in this work.

The API offers the characteristics of a Web Crawler and uses the principles of extracting information based on the Scraping technique, performing searches for the attack vectors present in phishing. Moreover, it has the function of performing searches for strings in the log files, inserting them in the database. API actions can be classified into three types: collecting information about URLs opened by users and updating the **url-database** collection, collecting information about e-mail messages received by users and verifying the **mail-database**, and building the collection **url-scrapy**.

Also, search engines can be seen as Python API modules which perform tasks specified in their fields. Figure 15 shows the organization of the collections used and their fields, for a better visualization of the structure.

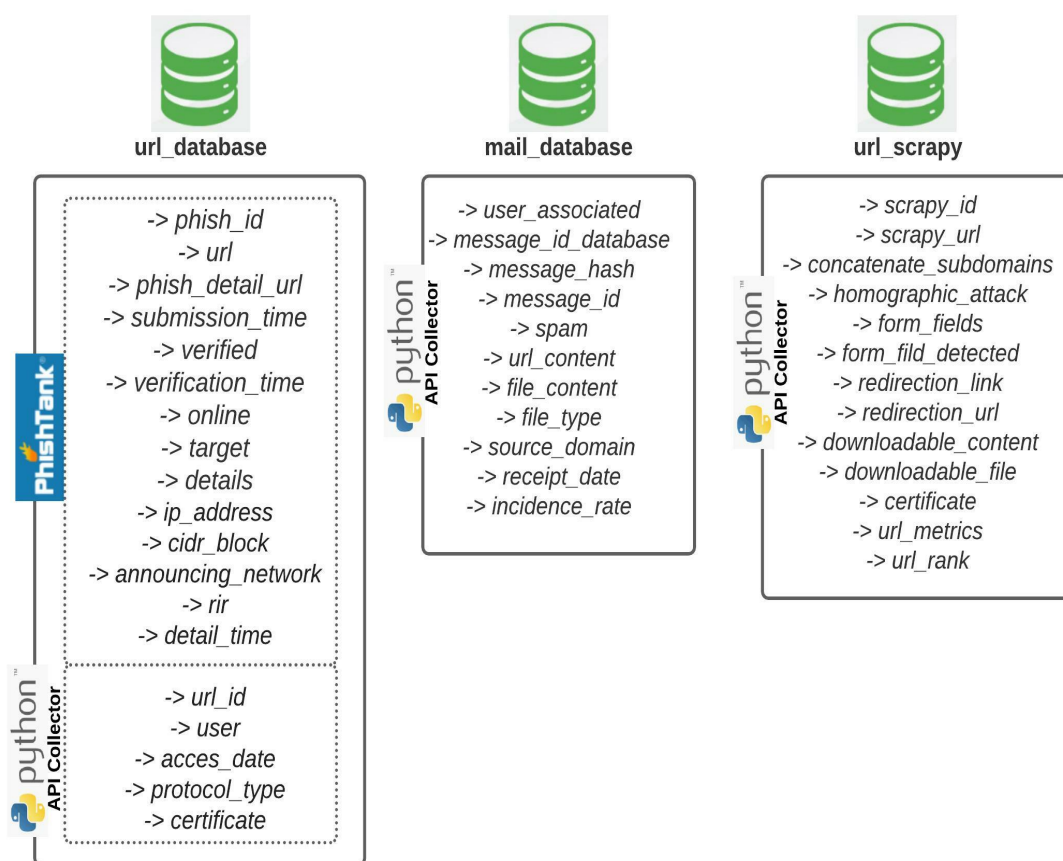


Figure 15 – Database Collections

The **url-database** collection is responsible for storing information concerning URLs, taken directly from the PhishTank base. The collection fields maintain the basic structure imported from the Phishtank database, preserving the characteristics of the import URLs such as:

- **phish\_id**: the identity (ID) number by which Phishtank refers to a phish submission. All data in PhishTank is linked to this ID. This will always be a positive integer.

- *phish\_detail\_url*: PhishTank detail URL for the phish, where it is possible to see data about the phish, including a screenshot and the community votes.
- *url*: the phish URL. This is always a string, and in the extensible markup language (XML) feeds may be a character data (CDATA) block.
- *submission\_time*: The date and time at which this phish was reported to Phish-tank.
- *verified*: whether or not this phish has been verified by community. In these data files, this will always be of the string 'yes' as only verified phishes are supplied in these files.
- *verification\_time*: the date and time at which the phish was verified as valid by the community.
- *online*: whether or not the phish is online and operational. In these data files, this will always be of the string 'yes' as only online phishes are supplied in these files.
- *target*: the name of the company or brand the phish is impersonating, if it is known.

Other fields are added to the collection ***url-database***, and accordingly are created and fed from the collection in the *log* files of the *Proxy*, from the URLs accessed by the user, and lastly, through the verification of the URL by the python API. Figure 16 shows an example of how the collection functions. The addition of these fields represents one of the contributions present in this study, whereby using *Strings* search engines, the developed application is able to detect important characteristics of the URLs:

- *url\_id*: URL id inside url-database captured via python API.
- *user*: the user who accessed the URL. This field is a composite field and can contain more than one user who accessed the same URL.
- *access\_date*: date/time of the access attempt.
- *protocol\_type*: brings the protocol present in the URL, hypertext transfer protocol (HTTP) or HTTPS.
- *certificate*: this field can be used in future investigations to develop new reliability metrics for digital certificates. In this current study, this field only presents the certificate used by the page if the return of the *protocol\_type* field is HTTPS.



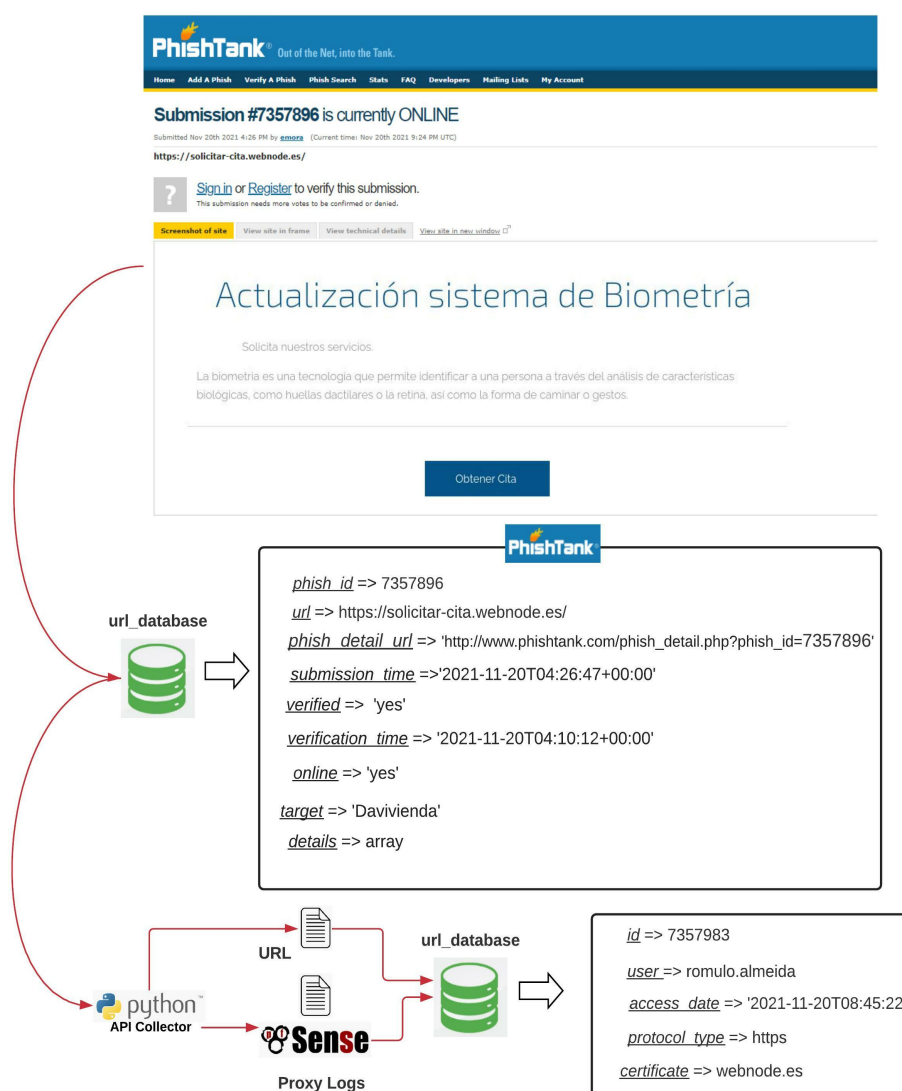


Figure 16 – Example of url-database collection

The **mail-database** collection stores information related to *email* messages and is fed directly from the *logs* of the *Anti Spam* tool (Figure 17). This collection corresponds to the *e-mail* checking method proposed in this work. Before initiating the user mailbox analysis process, the API Python makes a copy of the mailbox to the **mail-database**, consequently extracting the detected parsing objects and thus transforming them into the fields used by the collection. Accordingly, an input base for future comparisons is formed. This maneuver is performed message by message and requires some execution time. Some fields in the collection should be highlighted:

- *user\_associated*: indicates the user associated with the message.
- *message\_id\_database*: id of the parsed email on **mail-database**
- *message\_hash*: email message hash calculated via message authentication code

HMAC-MD5 by the python function (hashlib) implemented in the API.

- *message\_id*: id of the parsed email message.
- *spam*: the spam field is responsible for returning a YES or NO value as to whether the *email* message was detected or not as *Spam* by the *Anti Spam* tool. Later, this field will enable analysis of which *phishing* messages are being detected or not by the *Anti Spam* filter. By collecting this information, it will be possible to draw up studies on the detection rate of certain types of phishing messages.
- *url\_content*: contains information on whether the checking API detected a redirect *link* in the message body, and if so, which URL was found. For the detection of URLs, a string search metric is used in the message body, in an attempt to find words starting with http. An example might be: “ ^http+”.
- *file\_content*: contains information related to files attached to the message body which were detected by the API, returning a YES or NO; if YES, which file was found. As a means to detect files, a string search metric is used in the Content-Description field. This field may vary depending on the Anti Spam tool used. An example might be: “Content-Description: [file.extension]”.
- *file\_type*: returns the file extension found in the body of the analyzed message if the result of the *file\_content* field is YES.
- *source\_domain*: informs the domain that sent the message.
- *receipt date*: message receipt date.
- *incidence\_rate*: represents the message incidence rate, calculated through the Python API. This rate indicates whether the same message has been received before in the user’s mailbox. The incidence rate calculation is done through the python API using hash-based message authentication code with message digest algorithm 5 (HMAC-MD5) in the hashlib function (Figure 17). In this analysis, the message body is evaluated as part of the *source\_domain* field because the same message can be sent from different domains. In this analysis, the message body is evaluated as part of the *source\_domain* field because the same message can be sent from different domains. In this study, only the Message-ID, Subject, Date, Content-Type, and Content-Descriptions fields will be used to generate the *message\_hash* as they may vary depending on the Anti Spam tool used.

The message is collected from the Anti spam tool log in text form and transformed into a computational hash. This hash is stored in the *message\_hash* field. Whenever a new message is received, its hash is compared with that of other messages (*message\_id* + *message\_hash*). If the hash exists, an incidence rate will be given.

This rate can have the following values: 0 (the message has no impact on **mail-database**), 1-5 (the message appeared up to 5 times in **mail-database**), 5++ (the message appeared more than 5 times in **mail-database**).

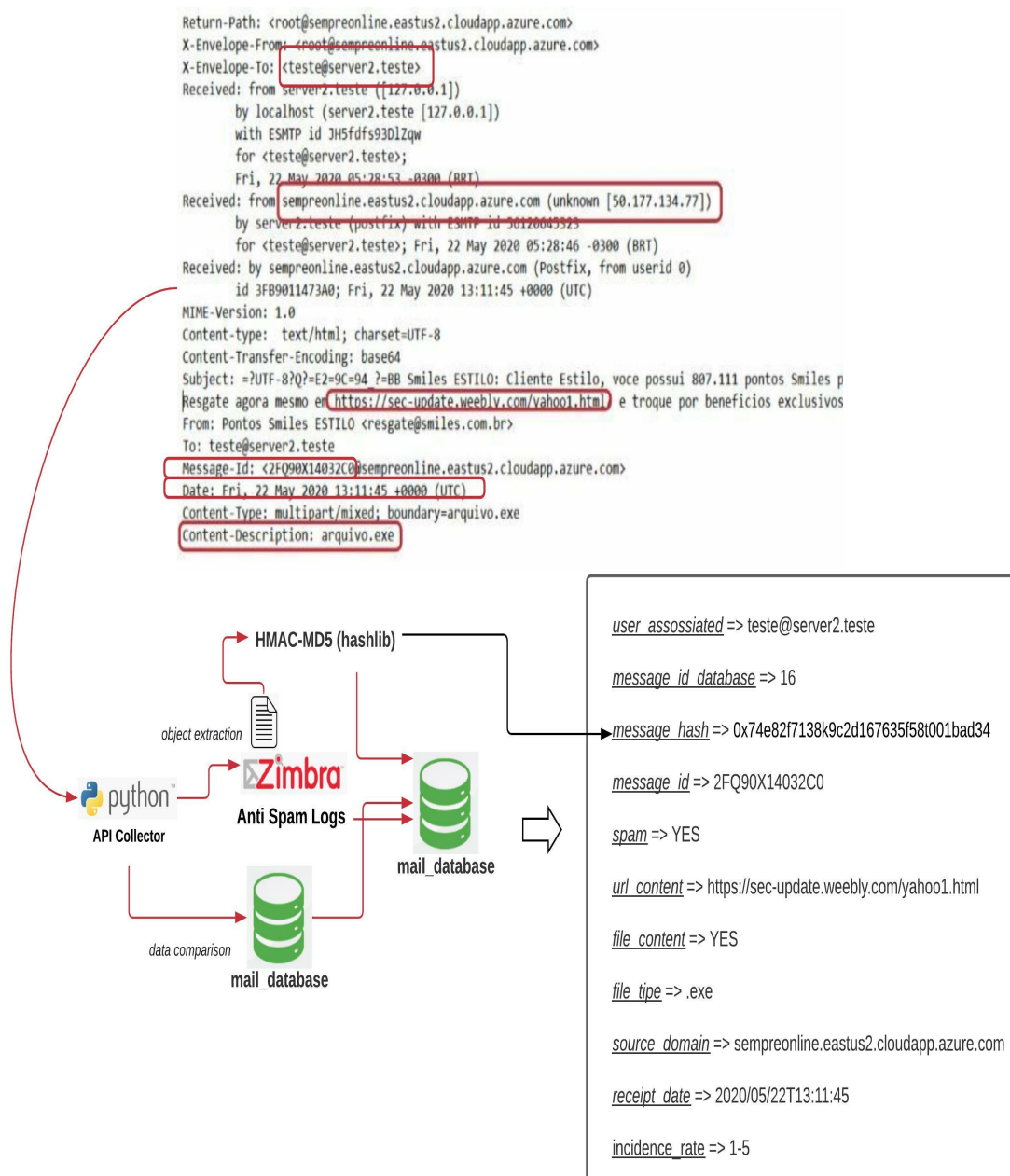


Figure 17 – e-mail collection example

The **url-scrapy** contains the values of the attack vector checks in detecting the characteristics of **phishing**, contributing part of the main proposal of this work. These elements analyzed are the main focus and proposal of this study because they represent the greatest link of computational interaction between its actions and fraudulent maneuvers. Consequently, these three characteristics are highlighted as the main attack

vectors present in phishing. In addition, other vectors will be analyzed: the concatenation of subdomains, homographic attack, and the absence of a digital certificate. These additional vectors are observed based on the work (SILVA et al., 2020). Analyzing these vectors through the web crawling technique performed by the python API (**url-scrapy**) will result in a phishing detection metric. In this step, a table (Figure 18) with the evaluated vectors will be presented.

[ ATTACK VECTOR ]	[ CLASSIFICATION GROUP ]	[ CRITICAL RATE ]	[ SCORE ]	[ SEARCH ENGINE MODULE ]	
Concatenate Subdomains <i>concatenate_subdomains</i> (SILVA et al., 2020)	Group A (URL Analysis) URL Morphology Method	Moderate	X	10	url_scrapy / <i>concatenate_subdomains</i>
		High			
Homographic Attack <i>homographic_attack</i> (SILVA et al., 2020)	Group A (URL Analysis) URL Morphology Method	Moderate		20	url_scrapy / <i>homographic_attack</i>
		High	X		
Certificate Missing <i>certificate</i> (SILVA et al., 2020)	Group A (URL Analysis) URL Morphology Method	Moderate	X	10	url_scrapy / <i>certificate</i>
		High			
Form Fields <i>(ALMEIDA;WESTPHALL,2020)</i>	Group B (Page Body Analysis) Web Crawling Method	Moderate	X	10	url_scrapy / <i>form_fields</i>
		High			
Redirection Links <i>redirection_link</i> (ALMEIDA;WESTPHALL,2020)	Group B (Page Body Analysis) Web Crawling Method	Moderate	X	10	url_scrapy / <i>redirection_links</i>
		High			
Downloadable Content <i>downloadable_content</i> (ALMEIDA;WESTPHALL,2020)	Group B (Page Body Analysis) Web Crawling Method	Moderate		20	url_scrapy / <i>downloadable_content</i>
		High	X		

Figure 18 – Table of Metrics - Classification of Attack Vectors and Score

In order to compose the final evaluation grade, these vectors will be divided into URL analysis (Group A) and page body analysis (Group B). In group A, the criteria related to the analyzed URL will be evaluated. The attack vectors presented here can be detected through the structure of the URL, and thus it is not necessary to analyze the elements in the body of the page and the HTTP headers. Subsequently, in group B, the objects contained in the body of the page will be evaluated. Consequently, a connection must be established and the elements will be read using the web crawling technique implemented by the API search engines.

Each Attack Vector can be classified into two severity levels (**moderate** and **high**) and their respective weights (**10**, and **20 points**). Vectors can appear individually and can also be combined into the two groups. A formula involving the combination of the vectors present and a margin of error will be used to calculate the phishing evaluation metric, resulting in a final grade. Analyzed URLs which meet the proposed grade will be considered suspected phishing.

Initially, a Full URL Trace is performed, analyzing the HyperText Markup Language (HTML) responses. This type of search functions well for classic websites or server-rendered pages where the HTML in the HTTP response includes all the content. If the use of Javascript, or any other dynamic language module on the page is detected, it will be executed step by step to detect all its resources (these criteria will be used to assess the complexity of the page). This action is necessary because the initial HTML content in some sites does not include the full content of the page itself, and thus, these are arranged inside small .js modules, asynchronous JavaScript and XML (AJAX) modules, JavaScript object notation (JSON), among others which are registered in the search engine.

The python API will queue all pages to be rendered unless a robot header or metatag directs not to index the page (Figure 19). It may be in this queue for a few seconds or more. Each chunk of .js code detected on a page will be executed and rendered sequentially, building the page's structure and storing it in memory. After reading all code blocks, the API will look for the structures found by links to other pages and save these links in the database. Previously accessed pages will already be cached through the previously performed rendering. Values are collected using the extraction API described in the next subsection. Some fields in the collection can be highlight :

- *scrapy\_url*: this field brings the URL evaluated by the web crawling module.
- *scrapy\_id*: id of URL scraped. This field will be used to distinguish when a URL arrived in url-database through import by Phishtank platform or when it entered url-database through web crawling module (*scrapy\_id* and *scrapy\_url*).
- *concatenate\_subdomains*: in this field, URL patterns which have multiple subdomain concatenations will be evaluated in order to make users believe, through a careless note, that the URL displayed in the browser is a legitimate domain (SILVA et al., 2020). An example of this occurrence can be seen in `acessoapp.login.bb2.com.br`. As reported in the experiments of the aforementioned study, it was possible to observe that using many subdomains are quite common invalid phishing attacks. To perform the analysis, the API search engine will consider a list of domains used in a whitelist and will count the dots (.) before or after the keyword (whitelisted domain). URLs with more concatenations will be considered suspicious.
- *homographic\_attack*: this field will evaluate the cases in which the fraudster makes use of substitutions, which may be misspelled words, paronyms, homographs, and homonyms, among others, which are often associated with word games that can pass by an inattentive end user (SILVA et al., 2020). Examples can be found in URLs using words like “faceb00k”, “Netfliix” or “Dr0pbox”. According to the

experiments carried out in the investigations above on Facebook, six substitutions were detected, totaling 8.07% of the samples. In addition, Brazilian banks, in particular "Banco do Brasil" also stands out with 4.17% of the total. This data led the attack vector to be considered highly relevant in phishing maneuvers. The feature was considered to be of high relevance.

- *form\_fields*: this field represents a YES or NO value and informs if form fields were detected on the analyzed page. The API search engine performs the search for form fields on the page (textfield). Initially, they will be registered commonly used string names and variables in addition to searching text fields like: 'username', 'login', 'user', 'code', 'number', 'card\_numer', 'mail', 'e-mail', 'key', 'security\_code', 'password' among others.
- *form\_fild\_detected*: returns form fields detected on the page by *form\_filds*.
- *redirection\_link*: this field represents a YES or NO value, stating whether the analyzed URL is redirected to other pages. As noted in (SILVA et al., 2020), the redirect evaluation will look for query string manipulation cases. The HTTP protocol allows the values of these parameters to be arbitrarily modified during the GET allowing a malicious user to place a malicious URL inside the legitimate URL which will redirect the user to a page that the fraudster entered in the parameters. The API search engine will also assess whether there are any other links embedded in the body of the page via .js modules or other types of dynamic languages.
- *redirection\_url*: if positive values are returned from the *redirection\_link*, a URL or redirect module (.js and others) will be collected.
- *downloadable\_content*: this field represents a YES or NO value regarding the existence of *download* content on the page. The Python API will also parse redirection to file-hosting links or self-running browser code used to call functions which perform auto-download on sites such as GitHub, among others. In addition, the tool will look for structures either in direct hosting links or front-end modules (.js) which automatically download files as the user clicks on a specific element of the page (an image or button on the screen which is not visible to the user but they will be downloading).
- *downloadable\_file*: if a positive value has been returned from *downloadable\_content*, the API would catalog the detected file and its extension.
- *certificate*: if the URL appears as HTTPS, it will contain its authentication certificate. Currently, even though it is known that many URLs of phishing have a

validated certificate, it is important to analyze which authority is signing the certificate. As reviewed in (SILVA et al., 2020) 88.85% of valid phishing sites do not use the HTTPS protocol, considering the vector as highly relevant for analyzing phishing pages. According to a 2017 survey for the current period, the number of valid phishing pages with HTTPS is growing, thereby highlighting the use of free encryption resources, such as Let's Encrypt.

The study also notes that with the increase in phishing pages with the secure protocol, it becomes questionable as to whether it should be considered a criterion to increase or decrease the reliability of the pages. Subsequently, this information can be useful to observe which certificates are mostly used by the fraudulent URLs and the comparison between certificates and valid domains.

- *url\_metrics*: this field represents the parameters used by the python API to calculate the URL evaluation metric. This metric will be detailed in subsection 3.1.3.
- *url\_rank*: this field represents the final score of the URL after being evaluated by the phishing detection metric proposed in this work.

```
2 # 1 - Verify for form fields in url
3 # 2 - Verify for downloadable contents in url
4 # 3 - Verify for links to another domains
5 # to run type $: scrapy runspider spider.py -s LOG_ENABLED=False
6 #import scrapy, for use this type $:pip install scrapy
7 import scrapy
8 class MySpider(scrapy.Spider):
9     name = "MySpider"
10    start_urls = [
11        #"https://anydesk.com/en"
12        #"https://google.com.br/"
13        "http://pudim.com.br/"
14    ]
15    def parse(self, response):
16        #Form Fields
17        form = response.xpath('//form').getall()
18        form_fields = False
19        if len(form) > 0:
20            form_fields = True
21
22        #Downloadable Content
23        download = response.xpath('//a[contains(@href, ".exe")]/@href').getall()
24        downloadable_content = False
25        downloadable_urls = []
26        if len(download) > 0:
27            downloadable_content = True
28            downloadable_urls = list(set(download))
29
30        #Redirect Links
31        redirect_links = response.xpath('//a[contains(@href, "http")]/@href').getall()
32        redirection_link = False
33        if len(redirect_links) > 0:
34            current_url = self.start_urls[0]
35            tmp = current_url[current_url.index('://')+2:]
36            current_domain = tmp[tmp.index('/'):]
37            redirect_links = list(set(redirect_links))
38            filtered_links = list(filter(lambda link: link.find(current_domain)
39                < 0, redirect_links));
40            if len(filtered_links) > 0:
41                redirection_link = True
42
43        #print results in console
44        print "Finish"
45        print "Url: ",self.start_urls[0]
46        print "Form_fields: ",form_fields
47        print "Downloadable content: ", downloadable_content
48        print "Downloadable urls: ", downloadable_urls
49        print "Redirect link: ", redirection_link
50        print "Redirect_urls: ", filtered_links
```

Figure 19 – Python API - Scraping Engines

### 4.1.3 URL score and rating metrics

For the elaboration of the metrics in order to evaluate the URLs, the Attack Vectors arranged in the work (SILVA et al., 2020) will be used, namely: Concatenate Subdomains, Homographic Attack, and Certificate Missing. In addition, the attack vectors proposed in (ALMEIDA; WESTPHALL, 2020) : Form Fields, Redirection Links, and Downloadable Content will also be added. A total of six attack vectors will be examined when evaluating the URLs; these vectors are extracted from the page through the Collector API using the web crawling technique and stored in the url\_scrapy collection.

Attack vectors will be scored by observing two criteria, incidence and criticality. The criticality can be seen in Figure 18, which can vary between "moderate" and "high" with values of 10 and 20. The incidence represents a relative value of the presence of the vector analyzed in the URLs found. As a result, a survey was conducted on the percentage of phishing URLs found which have each vector in (SILVA et al., 2020). Afterwards, a survey of the incidence of each attack vector in the URLs found in this work was taken. Figure 20 demonstrates the crossing of the incidence of vectors in the two studies. The results found in this work will be discussed in detail in chapter 5 (Experiment Results).

	ATTACK VECTORS DETECTED IN URLs (SILVA at al., 2020)	ATTACK VECTORS DETECTED IN URLs This Work
Concatenate Subdomains <i>concatenate_subdomains</i> (SILVA at al., 2020)	56,53%	49,31%
Homographic Attack <i>homographic_attack</i> (SILVA at al., 2020)	Percentage rated only on some URLs	80,71%
Certificate Missing <i>certificate</i> (SILVA at al., 2020)	88,85%	81,28%
Form Fields (ALMEIDA;WESTPHALL,2020)	Not rated in this work	66,16%
Redirection Links <i>redirection_link</i> (ALMEIDA;WESTPHALL,2020)	Not rated in this work	2,43%
Downloadable Content <i>downloadable_content</i> (ALMEIDA;WESTPHALL,2020)	Not rated in this work	16,83%

Figure 20 – Attack vector incidence percentage



From the incidence rates discovered in the two studies, a metric of values was created based on the incidence found (Figure 20). This Incidence Metric was calculated using intervals of 15%. The attack vector "downloadable content" appeared in 16.83% of the URLs, and consequently, it falls in the second interval of the Incidence Metric (16% - 31%). After fitting the attack vectors in the respective intervals, values were stipulated for them. These values were arranged considering the total number of analyzed vectors(6) and the maximum incidence percentage (100%), resulting in an approximate value of 16.66.

As the URL score will also be given as a Critical rate function (integer value), the incidence metric will be measured in decimal numerical form with a minimum value of **0.16**. Figure 21 shows the two metrics used and their ranges for the analyzed vectors. To elaborate the final formula, the following should be considered: **S** as the final note of the URL, **IR** the incidence rate of the detected attack vector, **CR** the critical rate of the detected attack vector, and **Q** the total number of attack vectors found. Given this information, the final URL score for the suspected phishing assessment can be calculated using the formula below:

$$S = \frac{(IRA1 * CRA1) + (IRA2 * CRA2) + \dots (IRAn * CRAn)}{Q}$$

With the pre-defined metric values (IR and CR) and the URL final score calculation formula, it is possible to infer a final average rating to determine whether or not the URL is suspected of phishing. The final average value of the chosen analysis is **6.4**. Analyzed URLs which have a final score below **6.4** will not be considered phishing by the evaluation algorithm, and URLs which obtain any score greater than or equal to 6.4 will be evaluated as phishing. The results discovered along with the observations on the accuracy of the metric will be discussed in chapter 5 (Experiment Results). Below, the proposed algorithm for analyzing URLs will be exemplified in pseudo-code form.

In summary, the algorithm input is the URL to be analyzed and tests the registered modules for each attack vector. These modules represent the search engines, based on web crawling which scan the page in search of the attack vectors mapped in this study. The results found are stored in the variable *RESULTS* and later in *Q*. If the value of *Q* is equal to zero, the URL will not be considered phishing as none of the attack vectors were found. If the value of *Q* is non-zero, **M-function** will be executed to calculate the URL grade. Before the execution of the **M-function**, a structure in matrix form  $M([i][j])$  containing three columns is stated: column zero with the name of each vector (RL, DC, CS...), column one with the **IR** of each vector (0.16, 0.32, 0.64...) and lastly column two, with the **CR** of each attack vector (10, 20, 10...). **The M-function** performs the search in the matrix arranged by the previously found modules in the algorithm and exhibits their results.

ATTACK VECTOR (A)	INCIDANCE %	% INTERVAL	INCIDANCE RATE (IR)	CRITICAL LEVEL	CRITICAL RATE (CR)	INDIVIDUAL SCORE (S)
Redirection Links (RL)	2,43%	0% - 15%	0,16	MODERATE	10	1,6
Downloadable Content (DC)	16,83%	16% - 31%	0,32	HIGH	20	6,4
-	-	32% - 47%	0,48	-	-	-
Concatenate Subdomains (CS)	49,31%	48% - 63%	0,64	MODERATE	10	6,4
Form Fields (FF)	66,16%	64 - 79%	0,80	MODERATE	10	8,0
Homographic Attack (HA)	80,71%	80 - 95%	0,96	HIGH	20	19,2
Certificate Missing (CM)	81,28%	80 - 95%	0,96	MODERATE	10	9,6
-	-	96% - 100%	1,12	-	-	-

Figure 21 – Attack vector incidence percentage

#### 4.1.4 API Execution and Collections Interactions

During the verification step of *url-database*, the API will search for the keyword in the log file, generated sequentially by a proxy, until it finds the string “*URL: [url]*”. This search done by the proposed search engine, captures a URL accessed by the user which was previously monitored by the content filter. After capturing this value, it will make a comparison with the respective collection contained in the database to detect if the URLs obtained are already inserted in the database. If it is found in the database, no action will be taken, as the URL has already been inserted into the database by importing the PhishTank Database.

Alternatively, the API performs a direct query to PhishTank in search of the URL found in the file read and then inserts it into the database together with the fields pre-determined in the request. If the URL acquired in the log file is not also present in the PhishTank, the API will run the test metrics registered in the search engines through the web crawling technique to detect if the URL has phishing characteristics. If the URL has such characteristics, it will be added to the *url\_database* with a *scrapy\_id*. The API will insert it into the database with a mark in the *new\_detected* field, indicating that

**Algorithm 1** URL evaluation algorithm

---

```

1: INPUT  $\leftarrow$  URL
2: Initialize RESULTS with the empty list of Arrays[].
3: Initialize MODULES with the list of Modules registered to verify each Attack vector (RL,DC,CS,FF,HA,CM)
4: for each module in MODULES ... do
5:   if
6:     The module was detected in the URL then
7:       RESULTS gets the list of Arrays with each module found in the URL
8:   end if
9: end for
10: Q  $\leftarrow$  RESULTS
11: if Q == 0 then
12:   The URL is not Phishing.
13: end if
14: if Q > 0 then
15:   for each result found in RESULTS ... do
16:     Execute the M-function for the RESULTS found and store in M
17:      $R \leftarrow R + M[IR] * M[CR]$ 
18:   end for
19:   SCORE  $\leftarrow$  R/Q
20:   MEDIA  $\leftarrow$  6.4
21:   if SCORE > MEDIA then
22:     The URL is suspected of Phishing
23:   end if
24:   if SCORE < MEDIA then
25:     The URL is not Phishing
26:   end if
27: end if

```

---

**Algorithm 2** M-function

---

```

1: INPUT  $\leftarrow$  RESULTS
2: Initialize M as the Matrix[i][j] of Attack Vectors (RL, DC, CS, FF, HA, CN) and their respective IR and CR.
3: for each modules in M ... do
4:   if
5:     The module was detected in M then
6:       return the module IRxCR
7:   end if
8: end for

```

---

a URL with characteristics of phishing was detected by the API, but was not added as suspicious on the platform.

After this, the API will report the URL to the Phishtank community as suspected phishing through the communication socket between the application and the platform. In this experiment, the tests performed are limited to the PhishTank database, not

preventing other databases from being coupled to the architecture in the future, thereby expanding the range of information. The Phishtank Platform allows the proposed API to perform queries in an integrated manner, thus facilitating the comparative framework in detecting URLs. Figure 22 provides a demonstration of how the collection and insertion of URLs are carried out.

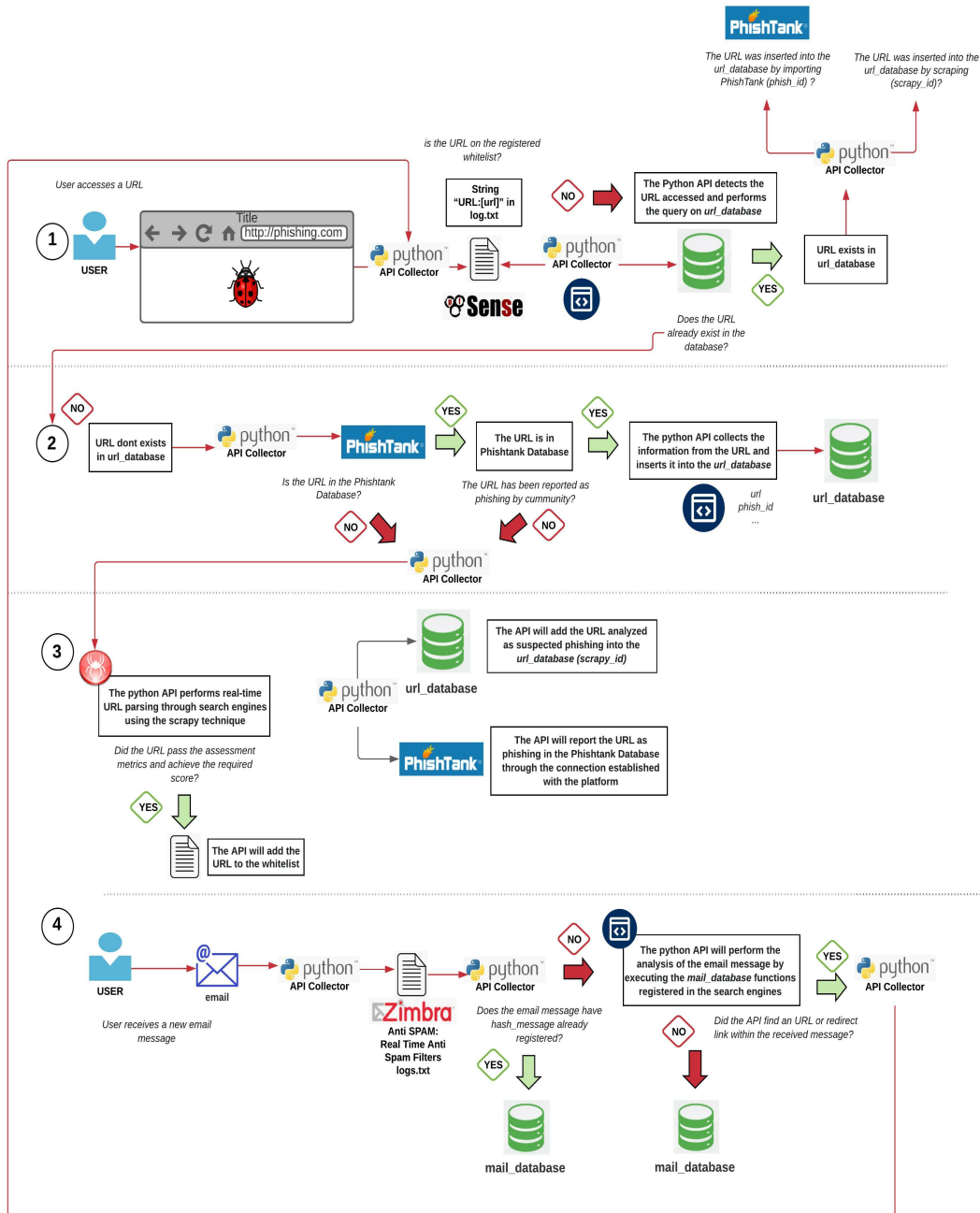


Figure 22 – API execution and collections interactions

Similar to the verification step `url_database`, upon collecting information for the `mail_database`, a new email message is received by the user and the API will read the file log generated by the Anti Spam tool. It will then perform the functions registered in the search engines related to `mail_database`. Before analyzing the message body, the

API will generate a hash of the message in HMAC-MD5 (*hashpy* Python function) to calculate the message incidence rate (*incidence\_rate*).

Thus, it is possible to know if the API previously analyzed the new message. The incidence rate is very common in phishing tricks and their derivations (spear phishing and whaling, among others). In addition to generating a message incidence rate, this mechanism manages to save API execution time, where a previously verified message will not be verified again. When the message body is the same, but the redirect link used in phishing is different, the *message\_hash* will be different, and the API will perform a new analysis. The ability to analyze the attached file type detects possible targeted malicious actions, in which malware is disguised as regular documents, such as spreadsheets and text documents.

## 4.2 USER BEHAVIOR MAPPING

This section is organized into four subsections. The first subsection presents the use of a taxonomy of the user's actions and the classification of the attack vectors used in social engineering and their respective threats and damages. This approach aims to elaborate a reduced taxonomic structure, delimiting the scope of exploration to the already known vectors (Figure 23). Subsection 4.2.2 presents the new additions made in the experimental scenario proposed in section 4.1. (new API's). Subsection 4.2.3 sets forth the implementation of the finite state machine (DFA) used to monitor user behavior. Finally, subsection 4.2.4 provides the elaboration of the user maturity cybersecurity profile tree through the proposed APIs.

### 4.2.1 Taxonomy of User Actions and Attack Vectors

Social engineering represents a fabricated trust founded between the entity causing the malicious action and the user, designed to collect information, commit fraud or gain access to confidential information. Through observation, a new taxonomic structure will be elaborated on in this stage of the study, which relates the attack vectors used in phishing maneuvers with the computational actions performed by the users when interacting with these vectors. The classified attack vectors are based on the three analysis objects proposed in (ALMEIDA; WESTPHALL, 2020), which were discussed in the previous section.

Attack vectors (**AV**) will be divided into two groups: malicious URLs (**AV1**) and malicious emails (**AV2**). Within each group, the attack vectors are divided into sub-vectors (**SB**). The AV1 group sub-vectors involve the presence of redirect links in URLs (**AV1SB1**), the presence of form fields in a URL (**AV1SB2**), and finally, the presence of download content on the page (**AV1SB3**). Users' computational actions (**UA**) will be classified as representing characteristic interactions in phishing maneuvers through

fraudulent URLs or malicious email messages. Figure 23 shows the proposed taxonomic structure.

The attack vectors were divided by observing the concepts presented in the literature, discussed in chapters 2 and 3 of this study. Both groups (**AV1 and AV2**) have characteristic steps present in executing the attacks. Phishing attacks emanate from social engineering ploys and digital espionage. In more specific phishing cases (spear-phishing and whaling), the collection of this information is usually carried out through passive internet scans (public information). The digital espionage process represents a large collection of information undetected by security devices, such as firewalls and IDS / IPS (Intrusion Detection System / Intrusion Prevention System).

This information can be collected through social network scans, domain lists, Google Dorks, social engineering applications such as Maltego and SET on Kali Linux, as well as other scans which seek information already indexed on the web (KOYUN; AL JANABI, 2017). Sending specific emails to a certain type of person with a mapped profile is also within the scope of targeted attacks. Generally speaking, phishing attacks can only involve hosting a malicious page (**AV1**) containing its attack sub-vectors (**AV1SB1, AV1SB2, and AV1SB3**), as in the case of fake pages from financial institutions and other services commonly used by the population in the digital medium.

To fully grasp the taxonomy used, some of the commonly observed situations involving phishing maneuvers and their derivations will be mentioned below. In addition, the types of damage caused by the incident will be listed as well.

1. **Situation 1:** The attacker publishes a phishing URL (**AV1**), pretending to be a financial institution, social networking page, or other service found on the Internet. This URL may or may not contain internal redirect links to other malicious pages (**AV1SB1**). The focus of the maneuver is the Theft of users' credentials through false trust and filling in form fields (**AV2SB2**). It then spreads it through social networks or other untrustworthy pages to reach users of these services in general, without specific targets. In addition, the phishing page can also be spread through a DNS poisoning attack.
  - **User Actions (UA):** The user somehow ends up finding the URL and clicking on it (**UA1AV1**). Users without the knowledge or with a low cyber-security maturity level can interact with form fields and Inform their credentials (**UA1AV1SB1 and UA1AV1SB2**). These credentials can range from simple data to credit card and bank account information.
  - **Threats Behind:** Common Phishing
  - **Type of Damage:** Total loss of information confidentiality. The credentials informed were lost and can be used to carry out other frauds on the internet.

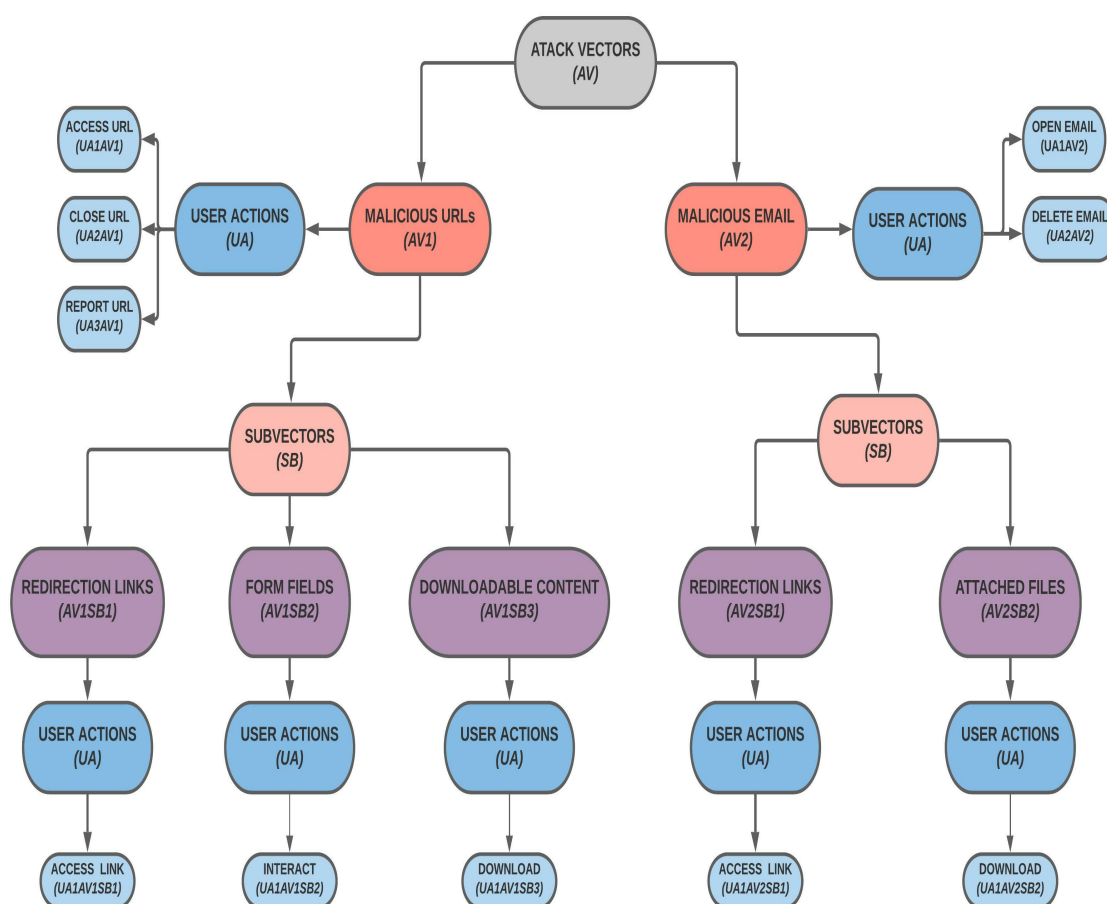


Figure 23 – Taxonomy of Attack Vectors and User Actions

2. **Situation 2:** The attacker performs a digital espionage maneuver through social engineering techniques and collects information about specific targets, such as personal or corporate email, social networks used, financial institutions used, among other services. The attacker then publishes a phishing URL (**AV1**) and spreads it via email (**AV2**) to specific targets previously mapped. This attack intends to steal credentials through the false credibility of pages through the filling in of form fields (**AV1SB1**, **AV1SB2** and **AV2SB1**).

- **User Actions (UA):** The user receives a malicious email in their inbox (**AV2**), often undetected by the Anti Spam tools. The user opens the message (**UA1AV2**) and clicks the redirect link contained in the body of the email (**UA1AV2SB1**) whereby they are taken to the phishing URL. Consequently, the user believes it to be a real website and thus passes their credentials through the form fields of the malicious page (**UA1AV1SB2** and **UA2AV1SB2**).
- **Threats Behind:** Common Phishing, Spear Phishing, Whaling, CEO Fraud.

- **Type of Damage:** Total loss of information confidentiality. The credentials informed were lost and can be used to carry out other frauds on the internet.

3. **Situation 3:** The attacker publishes a phishing URL (**AV1**) pretending to be a financial institution, social networking page, or other service found on the Internet. Inside the body of the page, a file is embedded for downloads such as internet banking files, vaccination calendars for COVID-19, and other file formats (.xls, .txt, .pdf) (**AV1SB3**). These files are generally of general interest to internet users. This URL may or may not contain internal redirect links to other malicious pages (**AV1SB1**) or form fields (**AV1SB2**). The form fields for credential theft also accompany most of the time, this type of maneuver.

The focus of the maneuver is infection through malicious code embedded in the files displayed (Ransomware, Trojans, and Keyloggers). Afterwards, the attacker spreads the URL through social networks or other untrustworthy pages to reach users of these services in general, without specific targets. In addition, the phishing page can also be spread through a DNS poisoning attack.

- **User Actions (UA):** The user somehow ends up finding the URL and clicking on it (**UA1AV1**). Usually on other ads/advertising pages with little credibility or through social media. Users with no knowledge or low maturity level in cybersecurity may interact with the form fields and enter their credentials (**UA1AV1SB1 and UA1AV1SB2**). In addition, users may download malicious files contained on the page (**UA1AV1SB3**). This type of activity is one of the main characteristics of malicious maneuvers involving movie hosting sites (torrent) and music in general. The file downloaded to the user's machine may look harmless, but when executed (**UA2AV1SB3**), it may invoke the malicious functions of obfuscated code within the structure of the file itself (Very common in office package files).
- **Threats Behind:** Common Phishing, Malware Infection, Command and Control, Information Hijacking / Data Encryption, Botnets.
- **Type of Damage:** Partial or total loss of integrity/availability of information.

4. **Situation 4:** The attacker performs a digital espionage maneuver through social engineering techniques and collects information about specific targets, such as personal or corporate email, used social networks, used financial institutions, among other services. In this case, the attacker can focus on a specific employee or employees of a company sector, for instance, an HR employee who receives resumes or a financial analyst who works directly with Excel payrolls. From there,



the attacker then publishes a phishing URL (**AV1**) and spreads it by email (**AV2**) to specific targets previously mapped.

This URL may contain a malicious file available for download on the Page (**AV1SB3**). In addition to the options mentioned above, the attacker can send a malicious file attached to the body of the email (**AV2SB2**); this file being something of interest to the previously mapped target (a fake resume for a recruiter or a cost sheet for someone in the financial sector). This attack is intended to infect specific targets by malicious code to implement other future attacks or the infection by malicious code embedded in the files displayed (Ransomware, Trojans, and Keyloggers).

- **User Actions (UA):** The user receives a malicious email in their inbox (**AV2**), usually undetected by anti-spam tools. The user opens the message (**UA1AV2**) and downloads the file attached to the email message (**AV2SB2**). The attached file was formulated specifically for the user's profile as it is a targeted attack for purposes other than phishing. The file downloaded to the user's machine may appear harmless, but when executed (**UA2AV2SB2**), it may invoke the malicious code obfuscation functions within the file structure (very common in office suite files).
- **Threats Behind:** Malware Infection, Command and Control, Information Hijacking / Data Encryption, Botnets.
- **Type of Damage:** Partial or total loss of integrity/availability of information.

Observing these characteristics, it is clear that the attacks use common exploratory routes, despite varying purposes and techniques. This analysis supports the hypothesis that attack vectors from social engineering used in phishing always have the same exploitation vectors, and even those with different techniques represent a limited number and are susceptible to detection/classification.

This line of analysis is fundamental for the second stage of this work, demonstrating that the problem of detection of characteristic attack vectors and the actions of users, or interaction with them, represents a finite problem. Thus, this problem can be computationally mapped by observing all the elements involved (**UA + AV**). When the attack vectors are mapped (**AV**), alongside the mapping of the behavior of each type of user (**UA**), the link between the beginning of the fraudulent maneuver and the execution of the attack can be detected in an organized way. Thus, when combined, the **UA** and **AV** groups become the way to conduct an incident.

#### 4.2.2 Additional Implementations in Experimental Scenario and new API modules

After the taxonomy substantiated above, the user behavior and the detected attack vectors can begin to be mapped. Attack vectors (**AV**) and their sub-vectors (**SB**) were already collected by the Collector API described in the previous section. The vectors referring to the URLs (**AV1**) are extracted through the application's search engines (Web Crawling) represented by the *url\_database* and *url\_scrapy* collections. Correspondingly, they go through the phishing detection metrics for the final URL score. The vectors referring to email messages (**AV2**) are evaluated by the Collector API and extracted by the implemented search engines represented by the *mail\_database* and *url\_database* collections.

With the aim of monitoring user behavior (**User Actions - UA**), two new modules will be proposed (API Behavior and API Automata), along with two new collections (*user\_behavior* and *system\_responses*). The new APIs and collections will be added to the experimental scenario proposed in the previous section (Figure 24). While conducting the experiments, the two APIs proposed on the users' machines were configured (following the agent model) under the Windows 10 operating system. In addition, a client/server communication was configured between the APIs on the machines and the Server Ubuntu, where the Mongo Database is located in port TCP 27017.

The Behavior API is responsible for monitoring the user's computational behavior. The API is installed as an agent on the user's machine and extracts information from the browser and operating system. For the experiments performed in this work, the API was built in two modules. The first was written in JavaScript language which performs direct communication with the Browser (Google Chrome), and the second, written in Python language which performs the core functions of the API and consumes the data from the JavaScript module. The JavaScript module is similar to a Google Chrome extension based on monitoring requests and meeting Chrome Developer Tool specifications. Figure 25 shows the minimum requirements needed to simulate the user environment.

In this first module, the API will monitor browser core events, such as: opening a new tab, closing a tab, starting a download, or other requests such as opening web sockets and clicking buttons in the body of the page. As such, it is possible to capture requests and monitor the life cycle of these requests, and it is also possible to interrupt them in case of improper actions during the process. This interaction allows you to monitor the entire operation of f12 (developer tools). The module also permits the monitoring of links which open with external links in other applications or programs, for example, a torrent link which requests opening directly in the torrent management application.

Furthermore, it is also possible to identify tabs which may be opened without the

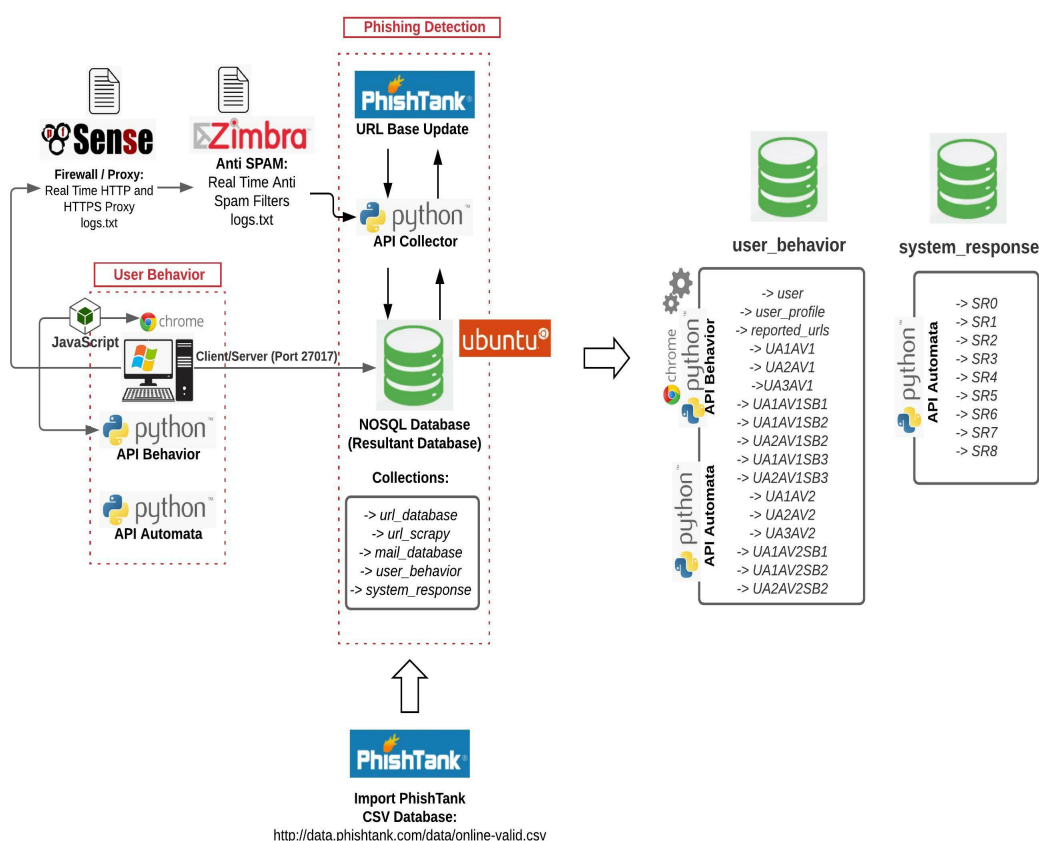



Figure 24 – New APIs and Additions

user's request by running applications in the background without the user's knowledge. Through this process, the API's JavaScript module reads the events that happen in the browser, managing to monitor the user actions (**UA's**) shown in Figure 24. The API Core, written in Python, consumes this information taken from the browser through the JavaScript bound extension and inserts them into the Mongo database.

The Automata API is responsible for preparing the user's maturity profile, basing its structure on monitoring user actions (**UA's**) and system responses (**SR's**) generated by API Behavior and API Collector. The creation of the user maturity profile will be discussed in section 4.2.4. The system responses are signals emitted by applications when confirming performed actions or parameter checks, and these signals need to be generated to be read by the Automata API and thus be part of the DFA alphabet. The **SR's** are generated when the actions take place and are related to the `time_check` field.

In addition, they are stored in the `system_response` collection. The `time_check` field is attributed to the moment when the Collector API and API Behavior checks occur, thus creating a relationship between the evaluated URL, performed user action, and the detection of this action. Furthermore, to build the Automata API, the `classDFA(FA) im-`

VIRTUAL MACHINE	OPERATING SYSTEM	VCPUs	MEMORY (GB)	DISK SPACE (GB)	IP
	Windows 10 or above	Core i5 or 8 VCPUs	8	240 SSD + 1024	172.16.30.100/24



**Considerations:** Use of DLLs and Python Libraries for Windows to Implement DFA and API\_Behavior - Both APIs must communicate with the MongoDB virtual machine (Port 27017) - Use of Chrome Developer Tools to implement the JavaScript module for capturing browser actions in Google Chrome

Figure 25 – User Virtual Machine Simulation

plemented in Python language through the automata-lib is used. Figure 25 shows basic descriptions of the configuration of the virtual machine (User's Machine) to reproduce the new implementation of APIs.

Below, some fields from the collection **user\_behavior** and **system\_responses** used by API Behavior and API Automata will be described.

- **UA1AV1:** API detects user action when opening a URL in a tab via browser.
- **UA2AV1:** The API detects the user action when closing a URL (tab already opened) through the browser.
- **UA3AV1:** API detects user action when reporting a URL as suspected phishing. For the reporting of suspicious URLs, a JavaScript function is configured in the API with a text field for entering the URL and a submit button. When reporting a URL, it will be recorded in the `url_reported` field and related to the user acting. This action is monitored to measure the maturity level of phishing URLs and generate a final analysis of which users could visually detect dangerous URLs. In this step, the `url_reported` field will be compared to the **url\_database** and **url\_scrapy** collection fields.
- **UA1AV1SB1:** The API detects user action when clicking a redirect link within a page.
- **UA1AV1SB2:** The API detects user action when typing information into form fields on a page. The API monitors the keyboard outputs introduced in the fields and sent by buttons (submit) by reading the requests in the extension coupled to the browser.

- *UA1AV1SB3*: The API captures the user action when downloading a file via the browser.
- *UA1AV2*: The API captures the user action when opening an email via the browser.
- *UA2AV2*: The API captures the user action when deleting an email via the browser.
- *UA1AV2SB1*: API detects user action when accessing URL within an email message via browser.
- *UA1AV2SB2*: The API detects the user's action when downloading a file attached to an email message via the browser.
- *SR0*: System response issued by API Behavior upon detection of user action when closing a URL or changing URLs (*UA2AV1*). The Automata API will understand this input as a restart of the automaton, taking it to the Initial state.
- *SR1*: System response issued by API Collector after checking URL score (not phishing/score < 6.4).
- *SR2*: System response issued by API Collector after checking the URL score (phishing/score > 6.4).
- *SR3*: System response issued by API Behavior upon detection of user action of clicking a redirect link within a malicious page (*UA1AV1SB1*).
- *SR4*: System response issued by API Behavior after detecting user action to fill a form field and/or interact with a submit button (*UA1AV1SB2*).
- *SR5*: System response issued by API Behavior after detecting user action when downloading a file via a URL in the browser (*UA1AV1SB3*).
- *SR6*: System response issued by API Collector after checking the email message opened by the user, where it was opened from the spam box.
- *SR7*: System response issued by API Collector after checking the email message opened by the user, where it was opened from the inbox.
- *SR8*: System response issued by API Behavior after detection of user action to delete the email message (*UA2AV2*). This input will be understood by API Automata as an automaton restart, taking to the Start state.

### 4.2.3 Finite State Machine Implementation

After describing the API Behavior and the new collections added, the Automata API will be proposed. The Automata API implements a finite deterministic automaton (DFA) responsible for monitoring user actions, the Behavior API collects browser actions, and the Automata API monitors the Behavior API and Collector API results. Therefore, the automaton has the user actions (UA) as input (alphabet) coupled with the system responses (SR), thus being able to trace the user's behavior profile.

The DFA alphabet is a finite alphabet, represented by the taxonomy proposed in Figure 23. At the end of the execution, it is possible to observe which DFA states the user went through as they performed their daily routines, thereby forming a maturity tree in the user's cybersecurity. Some states are more critical than others, demonstrating the user's degree of susceptibility to malicious maneuvers highlighted in this work.

The initial state of DFA represents the user starting normal tasks for the day, so this state is also a final state (acceptance) as it will only advance in the structure when the user performs any of the actions (UA) listed in the alphabet. Therefore, the proposed DFA will accept the empty entry ( $\epsilon$ ). Figure 26 shows the DFA in the form of a state diagram.

The B1 automata is constructed to read a group of strings composed of a string from the UA group followed by one or two strings from the SR group and can be represented in the 5-tuple form  $(Q, \Sigma, \delta, q_0, F)$  where:

- $Q$  is the finite set of states  $(S, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{r1}, q_{r2})$
- $\Sigma$  is the alphabet  $[\epsilon, UA1AV1, UA2AV1, UA3AV1, UA1AV1SB1, UA1AV1SB2, UA1AV1SB3, UA1AV2, UA2AV2, UA1AV2SB1, UA1AV2SB2, SR0, SR1, SR2, SR3, SR4, SR5, SR6, SR7, SR8]$ .
- $\delta$  is the transition function represented in the form:  $Q \times (\Sigma) \rightarrow Q$ .
- $S \in Q$  is the initial state.
- $F \subset Q$  is the set of accept states  $(S, q_{r1}, q_3, q_7, q_9, q_{12}, q_{13})$ .

DFA B1 starts in state  $S$ , where  $S$  is an automaton acceptance state. The transition to state  $q_1$  ( $\delta(S, UA1AV1) \rightarrow q_1$ ) is performed through the string  $UA1AV1$ , which represents the user action when opening a new URL in the browser, captured by the Behavior API. The transition  $\delta(S, UA1AV2) \rightarrow q_2$  will take from state  $S$  to state  $q_2$  via the  $UA1AV2$  entry and represents the user action when opening an email via the browser.

In state  $q_1$ , checking the URL through the Collector API will be performed. If the URL passes in the analysis metrics, the Collector API will generate the system

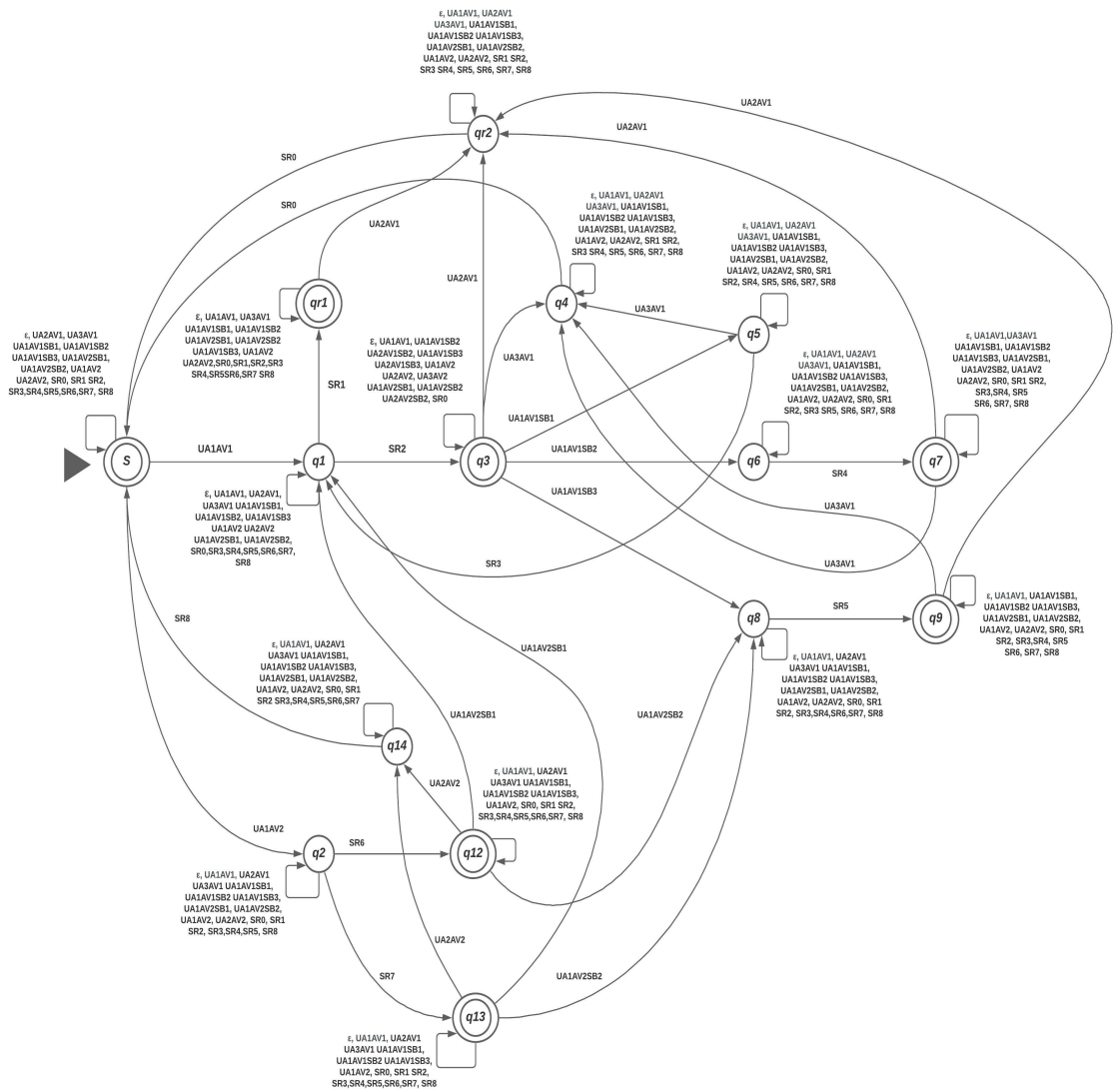


Figure 26 – DFA state diagram

response SR1, moving it to state qr1 through the transition  $\delta(q1, SR1) \rightarrow qr1$ . The qr1 state represents the user’s access to a URL that was not considered phishing, this URL can still represent other malicious features like a hoax and fake news, but these threats will not be evaluated in this work.

The qr1 state is an accept state as the user can still leave the URL open. The only input accepted by the state is the UA2AV1 entry (close browser tab/change URL) which will lead to qr2 state through transition  $\delta(qr1, UA2AV1) \rightarrow qr2$ . As a consequence of this user action (UA2AV1), the Behavior API will issue a system response SR0 which will take the automaton to the initial state S through the transition  $\delta(qr2, SR0) \rightarrow S$ .

Still in state q1, if the URL does not pass, does not reach the score required by the evaluation metrics (6.4), and is evaluated as phishing, the Collector API will issue a system response SR2, taking it to state q3 through the transition  $\delta(q1, SR2) \rightarrow q3$ .

The q3 state represents the user action when opening a URL in the browser

evaluated as phishing (Score > 6.4) through the checks carried out in the Collector API. This state represents the central point of the automaton and has the highest concentration of outputs within the structure. The  $\delta(q3, UA3AV1) \rightarrow q4$  transition occurs through the UA3AV1 entry, representing the user action in reporting the URL as phishing through the Behavior API interface. Consequently, the API Behavior will issue the SR0, which will take the DFA back to the initial state through the transition  $\delta(q4, SR0) \rightarrow S$ .

The transition  $\delta(q3, UA2AV1) \rightarrow qr2$  will lead to the state qr2, representing the user action to close the browser tab or change the URL in the same tab. Consequently, the API Behavior will issue the SR0, which will take the DFA back to the initial state through the transition  $\delta(qr2, SR0) \rightarrow S$ . The transition  $\delta(q3, UA1AV1SB1) \rightarrow q5$  represents the user action to click on a redirect link (UA1AV1SB1) within the suspicious URL. Therefore the URL Behavior will return to system response SR3 performing the transition  $\delta(q5, SR3) \rightarrow q1$  where the URL opened via the link will be re-evaluated. The  $\delta(q3, UA1AV1SB2) \rightarrow q6$  transition through the UA1AV1SB2 input occurs when the user fills in form fields present in the URL body and somehow submits them through a submit button.

As a consequence of this action, the API Behavior will issue a system response SR4 generating the transition  $\delta(q6, SR4) \rightarrow q7$  to state q7 (acceptance state). However, in state q7, the user may close the URL (UA2AV1) going to state qr2 through the transition  $\delta(q7, UA2AV1) \rightarrow qr2$  or the user may report the URL as phishing (UA3AV1) after realizing the interaction with a fraudulent page  $\delta(q7, UA3AV1) \rightarrow q4$ . Finally, the transition  $\delta(q3, UA1AV1SB3) \rightarrow qr8$  represents the user's action to download the file through interaction with the page (UA1AV1SB3). Consequently, the API Behavior will issue the SR5 taking the state q9 through the transition  $\delta(q8, SR5) \rightarrow qr9$ .

After performing the download, the user may notice that the URL is maliciously reporting it (UA3AV1), taking it to state q4 through the transition  $\delta(q9, UA3AV1) \rightarrow q4$  or closing the URL in the browser (UA2AV1), leading to state qr2 through the transition  $\delta(q9, UA2AV1) \rightarrow qr2$ .

The state q2 represents the user's action to open an email message through the browser (UA1AV2). Upon performing this action, the API Collector will perform the check and verify if the message was detected as spam or not. If the message is opened from the Spam box, the API issues a response from the SR6 system taking it to the final state q12 through the transition  $\delta(q2, SR6) \rightarrow q12$ . If the message is opened from the inbox and has not been detected as Spam, the Collector API issues a system response SR7 leading to the final state q13 through the transition  $\delta(q2, SR7) \rightarrow q13$ . This verification action is performed to enable a comparison of which users interact with messages already detected as Spam.

Furthermore, it is possible to measure the level of phishing detection, through the Anti Spam platform, by understanding which phishing messages are detected and



which are not. The state  $q_{12}$  represents an open message from the Spam and through the  $UA_{2AV2}$  entry can go to the final state  $q_{14}$  through the transition  $\delta(q_{12}, UA_{2AV2}) \rightarrow q_{14}$  when the user deletes the message. This check is conducted to know which users have a higher level of maturity, already realizing that a URL is malicious and thus deleting other interactions.

If the user clicks on any URL within the body of the email, or any existing redirect ( $UA_{1AV2SB1}$ ), a transition  $\delta(q_{12}, UA_{1AV2SB1}) \rightarrow q_1$  will take the automaton from state  $q_{12}$  to state  $q_1$ , executing as though the URL was opened in the browser. If the user downloads any file attached to the email message ( $UA_{1AV2SB2}$ ), the automaton will be moved from final state  $q_{12}$  to state  $q_8$  through the transition  $\delta(q_{12}, UA_{2AV2SB1}) \rightarrow q_8$ .

Scans performed using  $AV_2$  attack vectors allow differentiating when the user interacts with URLs on the WEB arriving from different sources of email messages and when it interacts with URLs and files arriving from email messages. The entries seen in  $q_{12}$  can be observed in state  $q_{13}$ , only differentiating that they are performed through the inbox (not detected as Spam).

#### 4.2.4 Maturity trees and user profile

In addition to monitoring user behavior, the Automata API has the enhancement of the user security maturity tree as its ultimate purpose. Accordingly, the Automata API implements a structure based on a binary tree which represents the states reached in the automaton and transcribes them in the nodes of the tree. As the user progresses through the structure and reaches the automata acceptance states, new nodes are created from the root (DFA Initial State  $S$ ).

In this experiment, the automaton rejection states are not used to avoid the loop. As such, the SRs are used by moving the automaton to specific states and, in some cases, restarting the structure. Practically, for every URL (browser tab) opened, DFA  $B_1$  is executed. The more tabs you open, the more times DFA  $B_1$  will be running. As the DFA waits for a UA or an SR to change state, it can be stopped to wait for the impulse, although this impulse may not happen. For example, in the case where the user opens a URL and does not perform any other action, and it remains open in the browser tab without being closed. This type of situation results in several open and unanswered `time_checks`, consequently interfering with creating the user's maturity tree, thereby generating incomplete trees.

A `TIME` function was implemented in the Behavior API code bypass this situation, which automatically inserts the next String for the automaton to move, always returning it to state  $S$  (the root of the tree). In this experiment, the time function is set to 10 minutes. We can highlight the user's situation with a URL detected as malicious and the reached state  $q_3$ . State  $q_3$  is an accept state, however, the automaton could advance

to states q4, q5, q6, and q8, if no action is taken. Performed by it, DFA B1 will be in a loop waiting for the next action (interact with the URL; close it or report it).

In order to avoid this situation, the TIME function of the API Behavior after 10 minutes without any action will automatically generate an SRA impulse (similar to SR0), ending the DFA and returning to the initial state S. As the TIME function is linked with `time_check`, if the user interacts in the same navigation tab again, the structure will be reassembled from the last acceptance state reached (q3), continuing the maturity tree. The time function is restarted for each impulse generated, whether it is a UA or an SR. Each maturity tree created will be stored in `user_profile` through traversal in order (left-root-right). The depth of the tree and the number of nodes are directly linked to the user's number of interactions (bad actions) and, consequently, to the states reached within the DFA.

The states qr1, qr2, q4, and q14 represent user security ripening actions, such as when a user notices a phishing URL and reports it or opens an email and notices the fraudulent message, thus deleting it. The nodes to the right of each subtree will represent the most mature user actions. Conversely, the nodes to the left of each subtree are the user's interactions with threats. Figure 27 demonstrates an example of a user maturity tree.

The user's maturity profile can be determined from the levels reached in the tree structure. In the initial structure proposed in this work, the nodes (leaves) on the left can reach up to three levels of depth (level 1, 2, and 3), the greater the depth level, the lower the cybersecurity maturity of the user. The maturity tree reflects the computational actions of the user when interacting with the threats mapped through the proposed taxonomy and monitored by the DFA, how closer the user gets to these threats, much more DFA states are reached and consequently more nodes on the left will be added to the tree. Cataloging user maturity levels can assist in cybersecurity awareness campaigns and threat intelligence engines.

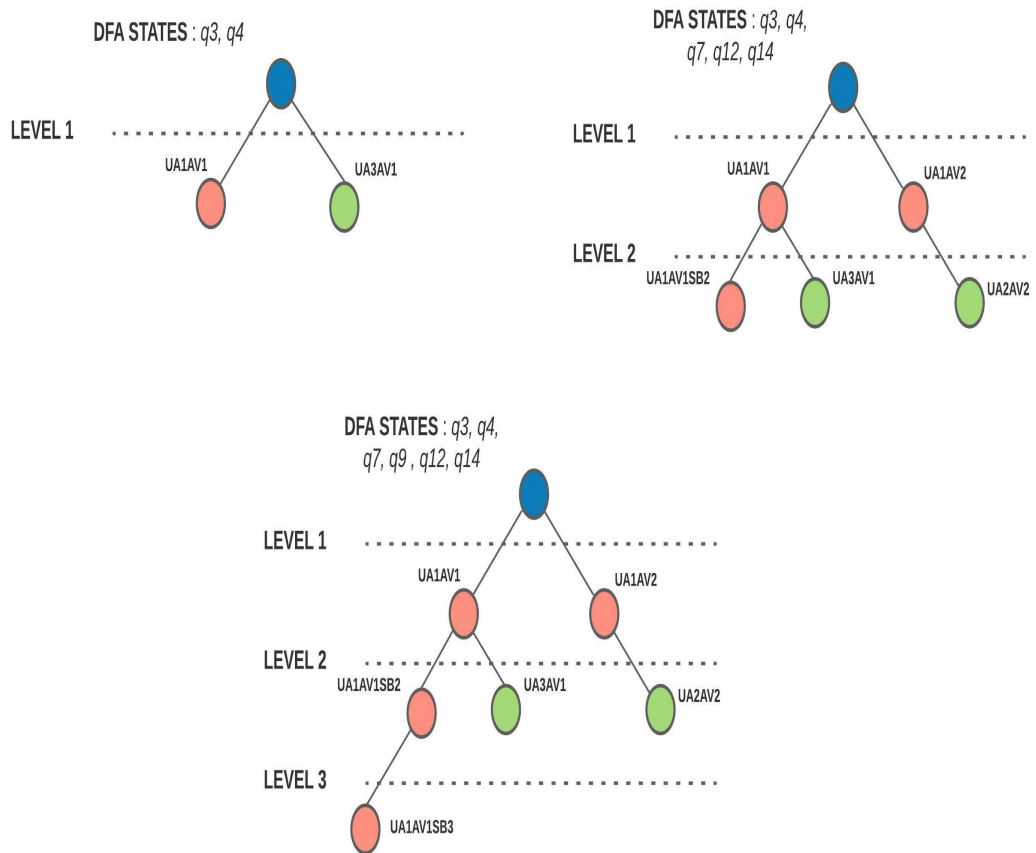


Figure 27 – Examples of User Maturity Tree

## 5 EXPERIMENTAL RESULTS

The monitoring of accessed URLs was undertaken between March 12th, 2020 to July 5th, 2020 among two hundred users, with an hourly frequency of data collection (collection of URLs in log files by the Scrapy API and synchronization with PhishTank). In total, approximately 18,000 URL samples were evaluated, where approximately only 12,000 URLs were able to be removed from the PhishTank Database and evaluated during the period. This was due to the fact that an approximate set of 6,000 URLs were no longer active or available for HTTP/HTTPS access.

In the end, the set of evaluated samples represented a total of 12,350 valid URLs, with approximately 12,000 URLs taken directly from the Phishtank base (counting updates every hour), and another 350 new URLs detected directly through the proposed methodology involving the URL collection `url_scrapy` through the Collector API. Of the 12,000 URLs exported from the Phishtank Database, approximately 84.81% of the URLs were elected by the community as phishing (10,178 URLs), of which 84.81% (8,796 URLs) were also recognized as phishing by the API. Figure 28 shows the comparative graphs of the API hit/error rate of the URLs taken from the Phishtank Database and collected directly from the users' access logs.

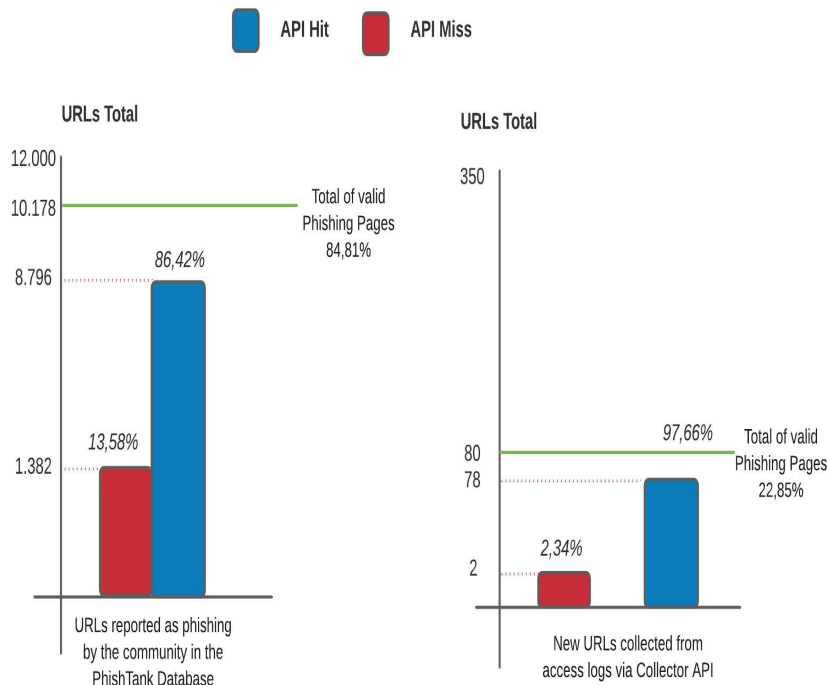


Figure 28 – Phishing Detection Accuracy - Hit/Miss

Only 13.58% of the URLs elected as phishing by the community (1,382 URLs) were not detected by the API. Notably, less than 3% of the 350 new URLs captured through the log files had also been reported as phishing in the Phishtank Database by

the community (10 URLs). All 10 URLs were also rated as phishing by the Collector API (Score greater than 6.4). Of the 350 new URLs, approximately 80 of these represented real phishing ploys. Of these, another 78 URLs were detected by the API and assessed as phishing, while only 2 URLs represented false negative. Subsequently, the API accuracy rate in detecting valid phishing URLs was 97.66%, with a margin of error of only 2.34%. The results gathered by the API Collector enable the possibility of visualizing how users kept in contact with phishing URLs. Figure 29 shows a comparative chart of valid URLs found vs. the PhishTank repository.

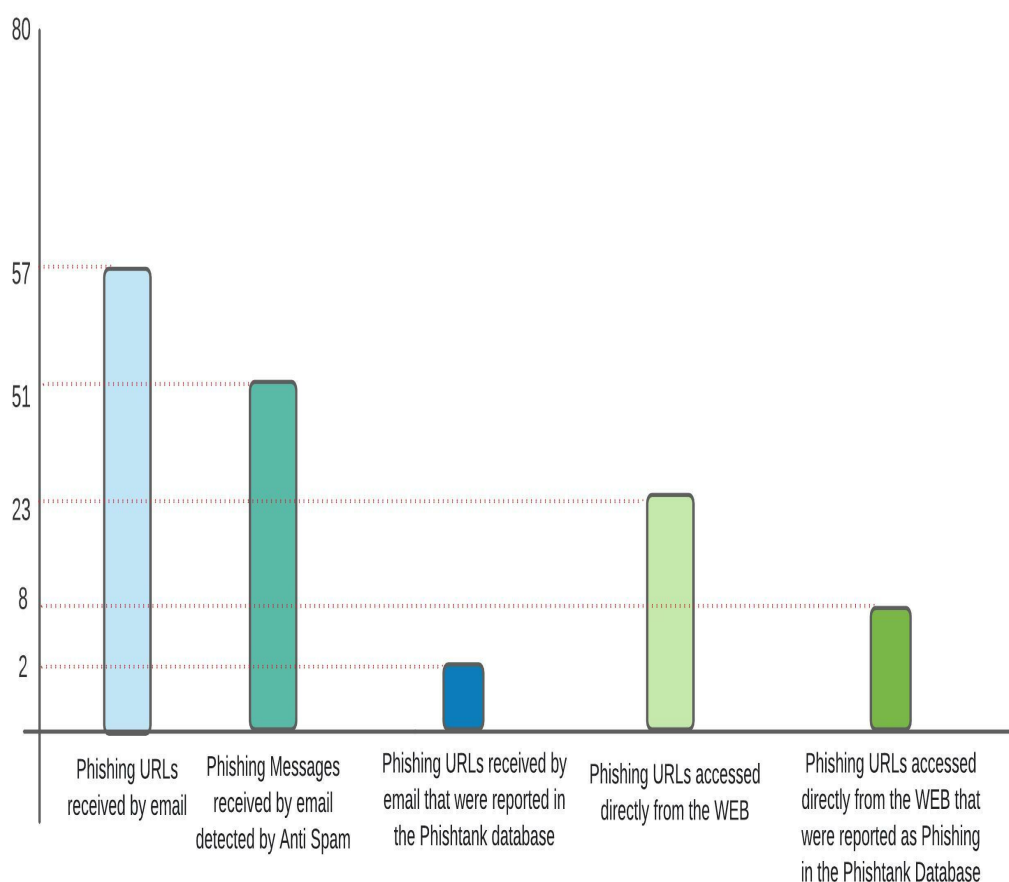


Figure 29 – Phishing Detection Accuracy - New Samples Vs PhishTank Database

It is worth noting the targeted phishing maneuvers (spear-phishing), in which users of e-mails such as *contracts@testdomain*, *recruitment@testdomain* and *financial@testdomain* received more phishing messages targeting their areas of activity compared to conventional phishing messages. These emails represent known accounts in the organization for open communication on the Web, such as receiving CVs for job vacancies or sending official documents and service contracts.

Attackers targeting more sophisticated and targeted social engineering maneuvers, perform web searches (footprinting) looking for these known boxes for sending targeted phishing messages/URLs (spear phishing). In addition, it was evident that

through scanning techniques the attackers detected which corporate tool was used to trigger emails (Zimbra), sending fake login pages to the highlighted accounts. The fake Zimbra platform login pages were accompanied by a notice that the corporate email service link had changed to the fake link (`home-zimb.firebaseio.com`). Malicious files, possibly containing ransomware or other types of malware, were sent in Microsoft Word document (`docx`) / portable document format (`pdf`) simulating resume files for job applications. Figure 30 provides an example of the phishing messages and URLs sent to these specific accounts.



Figure 30 – Phishing messages received via mailbox

At the end of monitoring malicious messages in the emails mentioned in the above boxes, approximately 70% were classified as spear phishing and 30 % as other types of phishing. In many e-mails, attackers simulated valid e-mails from other institutions or company employees requesting specific financial or contractual information. In some cases, only a file was attached, demonstrating that whale phishing and spear-phishing maneuvers do not always have classic phishing characteristics in fields and redirects.

This situation demonstrates the broad role of social engineering behind phishing maneuvers, leading to increasingly sophisticated attacks, and hence creating difficulty

to detect malicious actions through the proposed methodology. Through the incidence field of the Collector API, it was possible to detect repeated messages of spear-phishing and other types of phishing in the users' e-mail inbox. Generally speaking, the malicious messages which reached the maximum incidence rate (5+) were COVID-19 related messages and fake government emergency aid links. The message incidence rate was calculated by reading the Hash of the email message body through the API Collector. On many occasions, the messages had not been detected as Spam. Figure 31 shows examples of phishing messages with the highest incidence rate (5+).

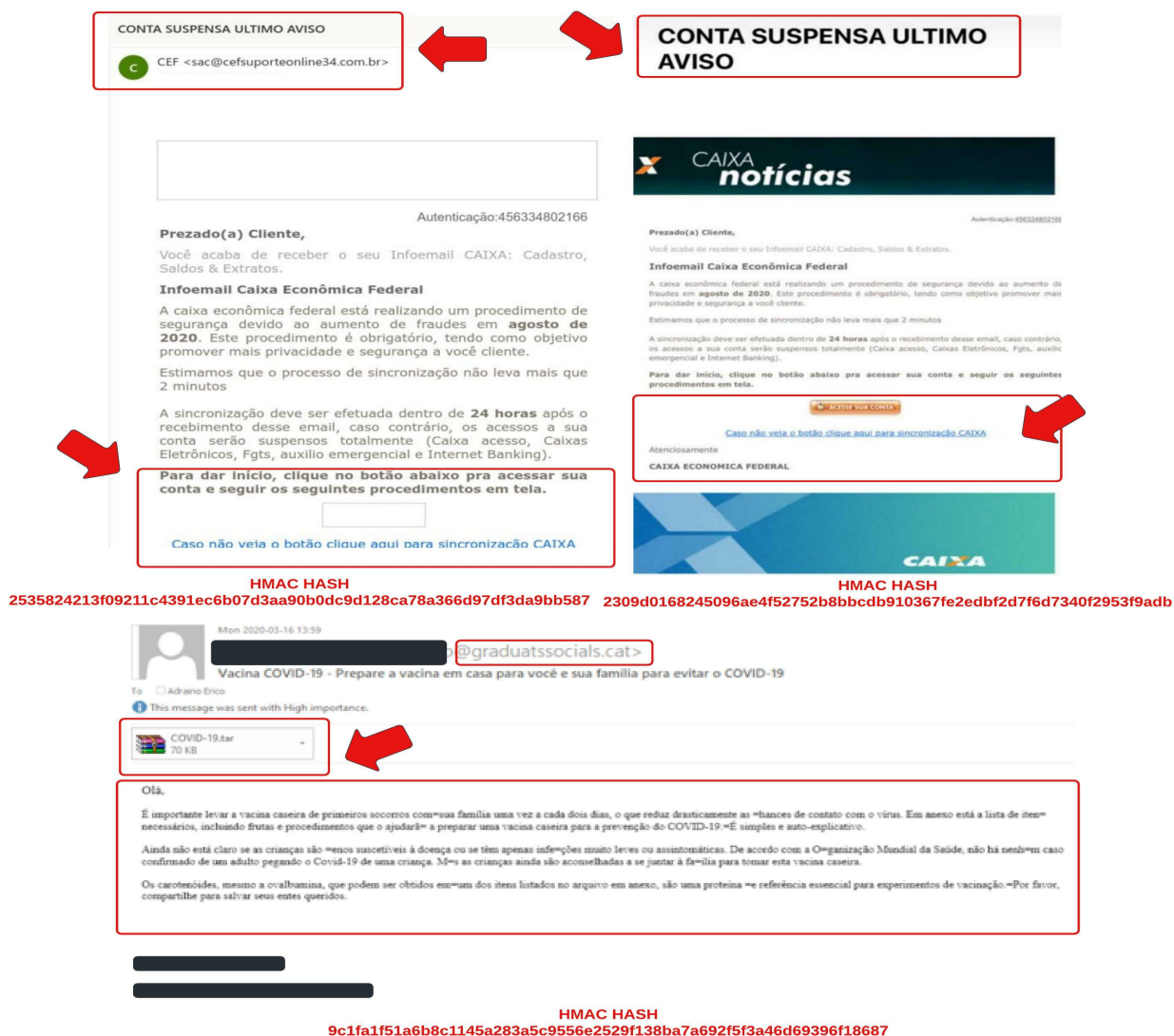


Figure 31 – Phishing messages received via mailbox - Pt2

The Hash calculation function achieved satisfactory results in detecting repeated phishing messages that contained the same images in the email body, regardless of the originating domains. Phishing messages related to COVID-19 were sent massively to test domain email accounts, along with pages related to government emergency relief.

In some boxes, these messages were received more than 10 times, with the same content as the body of the message (Fake image + redirect link).

During this period, it was identified that the main malicious URLs were clearly linked to the following themes: COVID-19, Help and government services, financial institutions, purchase and sale services (e-commerce), and airline ticket services. It is possible to characterize the malicious URLs into 6 groups with distinctly specific characteristics. Coronavirus-related URLs mostly lacked form fields but contained a suspicious file to download or redirect to other malicious pages, however most were not accessed from the mailbox. The links to services provided by the government mostly referred to false vaccination schedules and links to cash withdrawals. The financial assistance links all had form fields requesting user credit card data and credentials.

Furthermore, approximately 81% of the URLs did not have a digital certificate, and in more than 80% of the cases, they had the homographic attack as an attack vector to confuse users. Phishing groups related to financial institutions, as well as e-commerce buy and sell services, had characteristics similar to links connected to government campaigns and, in the vast majority of cases, were accessed through email messages in users' electronic mailboxes. Approximately 19% of these pages have a valid digital certificate, such as Let's Encrypt on their home pages, but in general, the pages which contained form fields to be filled in did not contain active encryption (HTTPS).

Concatenating subdomains were mainly highlighted on pages that requested login/password information from users on fake pages of banks and e-commerce services. Considering the eighty detected phishing URLs, the combination of the Homographic Attack (HA), Certificate Missing (CM), and Form Fields (FF) attack vectors appeared together in 57 URLs. Correspondingly, 71,25% of the New URLs phishing messages detected had this feature. Figure 32 shows the relationship of phishing pages found along with the incidence rate of attack vectors present in the URLs.

In regards to the checking methodology proposed in this article, the search engines showed varied results when reading the pages, depending on the layout and elements used in the pages' construction. On simplified pages (best case), as easily identifiable elements (HTML and cascading style sheets (CSS) without code omission), the application achieved an average of 97,66% accuracy in detecting form fields and redirect links with an average turnaround time of 12 seconds. Regarding the search for downloadable and executable files on the page, the average was 80% with a return of 20 seconds. Moreover, within the best scenario, the highest achieved values of accuracy and time were 97.66% and 15 seconds.

On more complex pages (worst case), with dynamic rendering elements (called JavaScript and code omissions), the algorithm used obtained a return greater than 5 minutes. The Scraping technique proved to be computationally costly in the interpre-



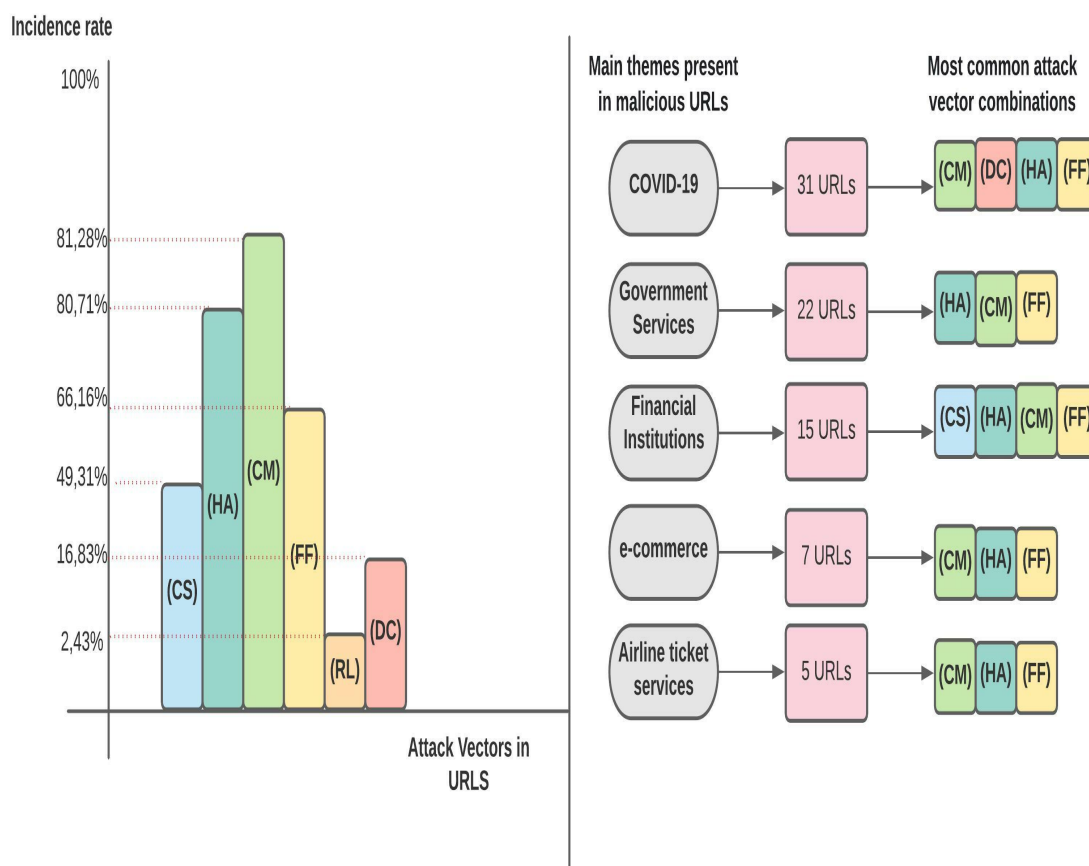


Figure 32 – Phishing occurrences

tation of URLs in search of phishing characteristics, requiring high processing power. The version of the algorithm proposed in this work is a prototype written in Python. It still needs improvements to deal with the fact that the vast majority of URLs do not have their elements available in a format which can be interpreted directly by the application. The evaluation metric proposed in this work proved to be efficient in detecting phishing pages, having only a 2% margin of error.

The monitoring of user behavior was observed in forty users across different sectors in a corporate setting. The users evaluated had different levels of knowledge in technology, and 10 of them had already gone through awareness campaigns and training in cybersecurity. It was observed that even when provided with a way to report malicious URLs through the browser, only two users reported URLs and denounced them as phishing. None of them had received training in cybersecurity.

A total of 6 URLs were reported as phishing. All of them had been detected by the Collector API as malicious. Some difficulties were observed during the execution of API Behavior, highlighting the execution time and generation of SRs for Automata API input. Consequently, in some situations, the DFA did not execute as expected. Therefore, the users' maturity trees were not generated. Altogether, approximately 210

maturity trees were created for the group of 40 users, of which only a set of fewer than 100 trees (without interference) brought consistent information for the elaboration of the profile. This group of 100 perfect trees made it possible to trace the continuous maturity profile in approximately 12 users. Within this profile evaluation, it was possible to observe which users accessed messages that had already been detected by the Anti Spam tool as malicious, which users filled form fields in malicious URLs, and also the users who downloaded malicious content from their workstations.

As expected, the phishing pages which saw the most interactions were spear-phishing pages that focused on specific email accounts (mostly accounts in the recruiting and contracting industry). During the analysis, a fake page of the federal government was also observed, specifically about COVID-19 which was fired out to all of the domains of the corporation. This page had form fields for registering and receiving the vaccination schedule. Of the 40 users, 34 filled out the form and sent information such as name, CPF, address, date of birth, among other information. Only six users closed the page as soon as they noticed the phishing maneuver, and only one user reported it as phishing through the browser JavaScript (API Behavior).

Some execution problems were detected, mainly related to the idle time of users while performing tasks, such as: opening a page in the browser and not performing any other functions. Even with the TIME function implemented, many trees were incomplete or looped and not completed. The Automata API generally had a response time for each alphabet entry and thus generated a tree node of approximately 10s. The longest collection time found in API Behavior while monitoring the UAs and SRs was 5 minutes. It was also possible to observe that among the group of 10 users with cybersecurity training, 7 had a lower maturity profile than other users without training and with little knowledge in technology. These trees which have reached the most critical states within the DFA are characteristic of phishing maneuvers aimed at strategic people within the corporation.

This result demonstrates that social engineering techniques, such as spear-phishing, go unnoticed even by users with more mature profiles in cybersecurity, thus being very difficult to be mitigated. Of the 12 complete maturity profiles generated, it was possible to map out all user actions (UAs) performed, demonstrating that the API Behavior, even with some functioning problems, presented itself as a viable way to capture computational actions of users. It was possible to classify the twelve generated profiles into three maturity levels (low, intermediate, and advanced) during the analysis of the twelve generated profiles. This classification made it possible to determine the maturity levels of the twelve evaluated users. Within this scope, seven users were classified as low level, reaching more critical states in the DFA and even taking actions which reached the final stages of executing phishing maneuvers or malicious code infection. Four users were classified as an intermediate level, and only one user was

classified with an advanced maturity profile.

## 6 CONCLUSIONS

This study has put forward a proposal for real-time phishing detection capable of detecting and analyzing malicious URLs based on heuristic techniques and web page scraping. An API was developed in Python language which runs its search engines to extract the main phishing characteristics from the pages: redirect links, form fields, content download, and other characteristics present in phishing maneuvers.

In order to evaluate the URLs, an algorithm was proposed which was written in Python and would perform the search for attack vectors in URLs using the web crawling technique. In addition, at this stage of the study, a metric to calculate the score of the analyzed URLs was also proposed, whereby if the URL had a value above 6.4, it would be evaluated as phishing. The proposed algorithm was implemented through the Collector API (described in chapter four) and reached an accuracy of 97.66% in detecting phishing pages. Furthermore, Collector API can check text files (firewall logs and Anti Spam filters) to search keywords such as URLs, domains, and attached files in email messages. This enables the application to be capable of detecting phishing tricks in email messages.

Finally, the API can also assess the incidence level of phishing messages in mailboxes by comparing the hash of the messages by the HMAC-MD5 (hashlib) function. Therefore, as made evident in the results found, it was possible to find phishing messages related to COVID-19 with an incidence rate greater than five points through the hash comparison. The message hash calculation method was implemented in its basic form and can be improved in future works for better accuracy in calculating the incidence of repeated phishing and spear-phishing messages in mailboxes.

Through the Collector API, it was also possible to analyze which malicious URLs have been going unnoticed by phishing reporting platforms, demonstrating cases in which fake pages were detected by the methodology proposed in this research and had not yet been evaluated as phishing on the PhishTank Platform, even though they were online for more than five days. The results obtained made it possible to observe that PhishTank Database did not play a relevant role in monitoring and reporting phishing pages in the Brazilian scenario.

Less than 3% of the application's valid phishing pages detected in real-time were on the platform. It is possible to infer that the PhishTank platform played a trivial role in monitoring phishing pages in PT-BR (Brazilian Portuguese) language for this work. Further investigations can be conducted on the efficiency of the platform regarding the PT-BR language and the Brazilian scenario of phishing URLs. Of the 18,000 URLs imported from the PhishTank Database, only 12,000 samples could be tested. The remaining samples were offline, rendering it impossible to check Collector API. Of the 12,000 URLs evaluated during the testing period, only a total of approximately

120 URLs were in the PT-BR language and corresponded to phishing pages linked to fraud in a Brazilian scenario. For URLs taken directly from the PhishTank platform, the proposed algorithm obtained an accuracy of 86.42% in detecting phishing pages, taking into account only those online URLs which had been reported as malicious by the community.

This study is language-independent and platform-independent for phishing evaluation compared to the works listed in chapter 3. In this work, the PhishTank platform was used, however other bases can be chosen or designed in derivatives such as an API collector, or even no initial base input could have been leveraged by performing a real-time capture of URLs and as obtained on phishing reporting platforms. In addition, the study's main highlight is the comparison to the complete evaluation of the content of the URLs found, consequently making it possible to adapt new attack vectors to the constructed algorithm.

The attack vectors suggested in this research may be changed, and new vectors may be coupled to search engines, providing versatility to the proposed detection method. The scraping technique proposed in this work has some limitations in extracting information from pages with more complex structures, which use other code structures embedded in the layout (JavaScript, JSON, among others). Larger, heavier pages took more than 60 seconds to return the extracted values, especially when checking redirect hyperlinks implemented through code calls within the pages.

Furthermore, the structure depends on a whitelist previously configured in the Firewall, containing a list of official URLs, thus avoiding many false positives and checking real pages. Through the URL comparison, which dispelled the homographic attack (HA), a whitelist of URLs and real domains was also used, where an API only performed a comparison. This method has several limitations, especially when there is a change in the body of access URLs for certain pages. In the future, new methods of detecting homographic attacks will be investigated which are computationally feasible to be coupled to the proposed algorithm's search engines. As future work will be extracted, new methods of improvement in search engines by scanning pages for phishing elements increase the detection of malicious URLs.

Another point to be investigated is making the API less dependent on security tools. While a proposed API behaves like a log reader and is compatible with other information security tools, third-party tools depend on it having specific log files. Lastly, there is also a possibility to investigate the feasibility of implementing only on the client-side, following state-of-the-art guidelines.

In addition to the phishing detection algorithm implemented through the Collector API, monitoring user behavior was also proposed in this work. This method was implemented through the Behavior and Automata APIs. The monitoring method is based on a taxonomy of computational user actions (UA) and attack vectors (AV) present in mali-

cious maneuvers, such as phishing and malicious code infection. This mapping enabled the elaboration of a structure capable of monitoring the user's computational behavior. As such, two application modules were installed on the users' machines. One of the Behavior API modules was created in JavaScript language and coupled to the browser (Google Chrome) based on the browser's Developer Tools. This module exhibited good results in capturing user interactions with pages accessed by the browser, detecting actions such as clicks and on-page interactive elements, opening tabs, closing tabs, filling out form fields and downloading files. These actions were mapped within the group of UAs.

The taxonomy of these actions played a fundamental role in developing this method of monitoring. Through the precise mapping of activities, it was possible to determine the exact moment in which a user came into contact with a malicious maneuver and whether the maneuver was implemented or not. This extraction method presented issues related to the processing time of activities in the browser. Some actions took more than five minutes to be cataloged, directly depending on available memory resources. The other module of the API Behavior was written in Python and installed on the client's machine and was only responsible for issuing the system responses (SR) to the Automata API.

In the future, more computationally viable ways to capture user actions through the browser will be investigated, thereby not creating high concurrency of computational resources. Another drawback observed was the users' idle time when performing actions in the browser, such as opening pages and leaving them on stand-by without performing other actions. A 10-minute TIME Function was implemented, which issues an SR to the Automata API, restarting the structure. The API Behavior was implemented in its prototype version to conduct the experiments present in this work, and, in the future, new methods to improve the algorithms used will be evaluated.

Ultimately, a structure based on a finite deterministic automaton (DFA), implemented through the Automata API in python, was used to monitor the environment. An alphabet based on the user actions (UA) performed and the system responses (SR) issued by the Behavior API served as the DFA input. The main difference of this method proposed, in comparison to the works listed in chapter 3, is the capture of actions performed by the user in real-time. Other studies have proposed behavior evaluation methodologies based on actions which have already taken place or evaluated residual samples in systems.

The proposed method in this study made it possible to develop profiles of users' cybersecurity maturity at the exact moment in which they were interacting with threats. During these experiments, it was possible to observe the advantage of using a DFA in regards to processing and memory resources. A significant challenge encountered in the proposed method involves the communication delay of capturing the actions from

the API Behavior from the browser to the Automata API. Using the proposed taxonomic structure and the DFA, it was possible to observe that the problem of mapping user actions and the characteristic attack vectors of phishing represent a finite problem, represented through a regular language. This proposition will lead this investigation into eventual future works on the feasibility of applying computational non-determinism in monitoring actions, which is applied through machine learning. Future investigations may be conducted into the advantages of using Machine Learning in both detecting phishing and monitoring user behavior.

## REFERENCES

- ABUZURAIQ, Almaha; ALKASASSBEH, Mouhammd; ALMSEIDIN, Mohammad. Intelligent Methods for Accurately Detecting Phishing Websites. In: IEEE. 2020 11th International Conference on Information and Communication Systems (ICICS). [S.l.: s.n.], 2020. P. 085–090. <https://doi.org/10.1109/ICICS49469.2020.239509>. DOI: 10.1109/ICICS49469.2020.239509.
- ADDAE, Joyce H; SUN, Xu; TOWEY, Dave; RADENKOVIC, Milena. Exploring user behavioral data for adaptive cybersecurity. **User Modeling and User-Adapted Interaction**, Springer, v. 29, n. 3, p. 701–750, 2019.
- AHMAD, Tabrez. Corona Virus (COVID-19) Pandemic and Work from Home: Challenges of Cybercrimes and Cybersecurity. **Available at SSRN 3568830**, 2020. <http://dx.doi.org/10.2139/ssrn.3568830>. DOI: 10.2139/ssrn.3568830.
- ALDAWOOD, Hussain; SKINNER, Geoffrey. An Advanced Taxonomy for Social Engineering Attacks. **International Journal of Computer Applications**, v. 177, n. 30, p. 1–11, 2020.
- ALHARTHI, Dalal N; HAMMAD, Mahmoud M; REGAN, Amelia C. A Taxonomy of Social Engineering Defense Mechanisms. In: SPRINGER. FUTURE of Information and Communication Conference. [S.l.: s.n.], 2020. P. 27–41. [https://doi.org/10.1007/978-3-030-39442-4\\_3](https://doi.org/10.1007/978-3-030-39442-4_3). DOI: 10.1007/978-3-030-39442-4\_3.
- ALJEAID, Dania; ALZHRANI, Amal; ALROUGI, Mona; ALMALKI, Oroob. Assessment of End-User Susceptibility to Cybersecurity Threats in Saudi Arabia by Simulating Phishing Attacks. **Information**, Multidisciplinary Digital Publishing Institute, v. 11, n. 12, p. 547, 2020.
- ALMEIDA, Rômulo; WESTPHALL, Carla. Heuristic Phishing Detection and URL Checking Methodology Based on Scraping and Web Crawling. In: IEEE. 2020 IEEE International Conference on Intelligence and Security Informatics (ISI). [S.l.: s.n.], 2020. P. 1–6.
- ANDRONIO, Nicolás; ZANERO, Stefano; MAGGI, Federico. Heldroid: Dissecting and detecting mobile ransomware. In: SPRINGER. INTERNATIONAL symposium on recent advances in intrusion detection. [S.l.: s.n.], 2015. P. 382–404.



BARRETT, MP. Framework for improving critical infrastructure cybersecurity. **National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep**, 2018.

BEAMAN, Craig; BARKWORTH, Ashley; AKANDE, Toluwalope David; HAKAK, Saqib; KHAN, Muhammad Khurram. Ransomware: Recent Advances, Analysis, Challenges and Future Research Directions. **Computers & Security**, Elsevier, p. 102490, 2021.

BISHOP, Matt et al. **A taxonomy of unix system and network vulnerabilities**. [S.l.], 1995.

CARLTON, Melissa; LEVY, Yair; RAMIM, Michelle. Mitigating cyber attacks through the measurement of non-IT professionals' cybersecurity skills. **Information & Computer Security**, Emerald Publishing Limited, 2019.

CHIEW, Kang Leng; CHOO, Jeffrey Soon-Fatt; SZE, San Nah; YONG, Kelvin SC. Leverage website favicon to detect phishing websites. **Security and Communication Networks**, Hindawi, v. 2018, 2018. <https://doi.org/10.1155/2018/7251750>. DOI: 10.1155/2018/7251750.

CIALDINI, Robert B; JAMES, Lloyd. **Influence: Science and practice**. [S.l.]: Pearson education Boston, MA, 2009. v. 4.

CICALA, Fabrizio; BERTINO, Elisa. Analysis of Encryption Key Generation in Modern Crypto Ransomware. **IEEE Transactions on Dependable and Secure Computing**, IEEE, 2020.

DHAMIJA, Rachna; TYGAR, J Doug; HEARST, Marti. Why phishing works. In: PROCEEDINGS of the SIGCHI conference on Human Factors in computing systems. [S.l.: s.n.], 2006. P. 581–590.

DIAZ, Alejandra; SHERMAN, Alan T; JOSHI, Anupam. Phishing in an academic community: A study of user susceptibility and behavior. **Cryptologia**, Taylor & Francis, v. 44, n. 1, p. 53–67, 2020.

DOBOLYI, David G; ABBASI, Ahmed. Phishmonger: A free and open source public archive of real-world phishing websites. In: IEEE. 2016 IEEE Conference on Intelligence and Security Informatics (ISI). [S.l.: s.n.], 2016. P. 31–36. <https://doi.org/10.1109/ISI.2016.7745439>. DOI: 10.1109/ISI.2016.7745439.

GOSTEV, Alexander. Kaspersky security bulletin. **Statistics**, p. 1–26, 2020.

GUPTA, Brij B; ARACHCHILAGE, Nalin AG; PSANNIS, Kostas E. Defending against phishing attacks: taxonomy of methods, current issues and future directions. **Telecommunication Systems**, Springer, v. 67, n. 2, p. 247–267, 2018.

HADLINGTON, Lee. Human factors in cybersecurity; examining the link between Internet addiction, impulsivity, attitudes towards cybersecurity, and risky cybersecurity behaviours. **Heliyon**, Elsevier, v. 3, n. 7, e00346, 2017.

HOWARD, John D; LONGSTAFF, Thomas A. **A common language for computer security incidents**. [S.I.], 1998.

ISKHAKOV, Andrey Yunusovich; ISKHAKOVA, Anastasia Olegovna; MESCHERIAKOV, Roman Valerievich; BENDRAOU, Reda; MELEKHOVA, Ol'ga N. Application of user behavior thermal maps for identification of information security incident., - ..., v. 61, n. 0, p. 147–171, 2018.

JAGATIC, Tom N; JOHNSON, Nathaniel A; JAKOBSSON, Markus; MENCZER, Filippo. Social phishing. **Communications of the ACM**, ACM New York, NY, USA, v. 50, n. 10, p. 94–100, 2007.

JAIN, Ankit Kumar; GUPTA, Brij B. Two-level authentication approach to protect from phishing attacks in real time. **Journal of Ambient Intelligence and Humanized Computing**, Springer, v. 9, n. 6, p. 1783–1796, 2018.

<https://doi.org/10.1007/s12652-017-0616-z>. DOI: 10.1007/s12652-017-0616-z.

JAMES, Lance. **Phishing exposed**. [S.I.]: Elsevier, 2005.

KHOUSSAINOV, Bakhadyr; NERODE, Anil. **Automata theory and its applications**. [S.I.]: Springer Science & Business Media, 2012. v. 21.

KISSEL, Richard. **Glossary of key information security terms**. [S.I.]: Diane Publishing, 2011.

KOYUN, Arif; AL JANABI, Ehssan. Social engineering attacks. **Journal of Multidisciplinary Engineering Science and Technology (JMEST)**, v. 4, n. 6, p. 7533–7538, 2017.

KROMBHOLZ, Katharina; HOBEL, Heidelinde; HUBER, Markus; WEIPPL, Edgar. Advanced social engineering attacks. **Journal of Information Security and applications**, Elsevier, v. 22, p. 113–122, 2015.

KRSUL, Ivan; SPAFFORD, Eugene; TRIPUNITARA, Mahesh, et al. Computer vulnerability analysis. **COAST Laboratory, Purdue University, West Lafayette, IN, Technical Report**, 1998.

LAVION, Didier et al. PwC's Global Economic Crime and Fraud Survey, 2020. **URL: <https://www.pwc.com/gx/en/forensics/global-economic-crime-and-fraud-survey-2020.pdf>** (retrieved 28 February 2020), 2020.

LINDQVIST, Ulf; JONSSON, Erland. How to systematically classify computer security intrusions. In: IEEE. PROCEEDINGS. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097). [S.l.: s.n.], 1997. P. 154–163.

LÓPEZ, Alberto Urueña; MATEO, Fernando; NAVIO-MARCO, Julio; MARTINEZ-MARTINEZ, José Maria; GÓMEZ-SANCHIS, Juan; VILA-FRANCÉS, Joan; SERRANO-LÓPEZ, Antonio José. Analysis of computer user behavior, security incidents and fraud using Self-Organizing Maps. **Computers & Security**, Elsevier, v. 83, p. 38–51, 2019.

MARTIN, Jaclyn; DUBÉ, Chad; COOVERT, Michael D. Signal detection theory (SDT) is effective for modeling user behavior toward phishing and spear-phishing attacks. **Human factors**, SAGE Publications Sage CA: Los Angeles, CA, v. 60, n. 8, p. 1179–1191, 2018.

MASHIANE, Thulani; KRITZINGER, Elamarie. Theoretical Domains Framework Applied to Cybersecurity Behaviour. In: SPRINGER. COMPUTER Science On-line Conference. [S.l.: s.n.], 2020. P. 411–428.

MITNICK, Kevin D; SIMON, William L. **The art of deception: Controlling the human element of security**. [S.l.]: John Wiley & Sons, 2003.

MOHEBZADA, Jamshaid G; EL ZARKA, Ahmed; BHOJANI, Arsalan H; DARWISH, Ali. Phishing in a university community: Two large scale phishing experiments. In: IEEE. 2012 international conference on innovations in information technology (IIT). [S.l.: s.n.], 2012. P. 249–254.

NARWAL, Bhawna; MOHAPATRA, Amar Kumar; USMANI, Kaleem Ahmed. Towards a taxonomy of cyber threats against target applications. **Journal of Statistics and Management Systems**, Taylor & Francis, v. 22, n. 2, p. 301–325, 2019.

OLSTON, Christopher; NAJORK, Marc. **Web crawling**. [S.l.]: Now Publishers Inc, 2010.

OVELGÖNNE, Michael; DUMITRAȘ, Tudor; PRAKASH, B Aditya; SUBRAHMANIAN, VS; WANG, Benjamin. Understanding the relationship between human behavior and susceptibility to cyber attacks: A data-driven approach. **ACM Transactions on Intelligent Systems and Technology (TIST)**, ACM New York, NY, USA, v. 8, n. 4, p. 1–25, 2017.

PARK, Andrew J; QUADARI, Ruhi Naaz; TSANG, Herbert H. Phishing website detection framework through web scraping and data mining. In: IEEE. 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). [S.l.: s.n.], 2017. P. 680–684.

<https://doi.org/10.1109/IEMCON.2017.8117212>. DOI:

10.1109/IEMCON.2017.8117212.

PRADO, Neriberto; PENTEADO, Ulisses; GRÉGIO, André. Metodologia de detecção de malware por heurísticas comportamentais. **Simpósio Brasileiro em Segurança da Informação e de Sistemas. ISSN**, p. 2176–0063, 2016.

RAMLO, Susan E; NICHOLAS, John B. Divergent student views of cybersecurity. **Journal of Cybersecurity Education, Research and Practice**, v. 2019, n. 2, p. 6, 2020.

RAO, Routhu Srinivasa; PAIS, Alwyn Roshan. Jail-Phish: An improved search engine based phishing detection system. **Computers & Security**, Elsevier, v. 83, p. 246–267, 2019. <https://doi.org/10.1016/j.cose.2019.02.011>. DOI:

10.1016/j.cose.2019.02.011.

SADQI, Yassine; MALEH, Yassine. A systematic review and taxonomy of web applications threats. **Information Security Journal: A Global Perspective**, Taylor & Francis, p. 1–27, 2021.

SAHINGOZ, Ozgur Koray; BUBER, Ebubekir; DEMIR, Onder; DIRI, Banu. Machine learning based phishing detection from URLs. **Expert Systems with Applications**,

Elsevier, v. 117, p. 345–357, 2019. <https://doi.org/10.1016/j.eswa.2018.09.029>. DOI: 10.1016/j.eswa.2018.09.029.

SALAH DINE, Fatima; KAABOUC H, Naima. Social engineering attacks: A survey. **Future Internet**, Multidisciplinary Digital Publishing Institute, v. 11, n. 4, p. 89, 2019.

SANTOS, Welton; FAZZION, Elverton; FONSECA, Osvaldo; CUNHA, Ítalo; CHAVES, Marcelo HPC; HOEPERS, Cristine; STEDING-JESSEN, Klaus; GUEDES, Dorgival; MEIRA JR, Wagner. Uma Metodologia para Agrupamento e Extraç ao de Informaç oes de URLs de Phishing. In: BRAZILIAN Symposium on Information and Computational Systems Security. [S.l.: s.n.], 2019.

SHARMEEN, Shaila; AHMED, Yahye Abukar; HUDA, Shamsul; KOÇER, Bari Ş; HASSAN, Mohammad Mehedi. Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches. **IEEE Access**, IEEE, v. 8, p. 24522–24534, 2020.

SHENG, Steve; HOLBROOK, Mandy; KUMARAGURU, Ponnurangam; CRANOR, Lorrie Faith; DOWNS, Julie. Who falls for phish? A demographic analysis of phishing susceptibility and effectiveness of interventions. In: PROCEEDINGS of the SIGCHI conference on human factors in computing systems. [S.l.: s.n.], 2010. P. 373–382.

SILVA, Carlo Marcelo Revoredo da; FEITOSA, Eduardo Luzeiro; GARCIA, Vinicius Cardoso. Heuristic-based strategy for Phishing prediction: A survey of URL-based approach. **Computers & Security**, Elsevier, v. 88, p. 101613, 2020.

SIPSER, Michael. Introduction to the Theory of Computation. **ACM Sigact News**, ACM New York, NY, USA, v. 27, n. 1, p. 27–29, 1996.

SOUZA, Cristian Henrique M; LEMOS, Marcilio OO; SILVA, Felipe S Dantas; ALVES, Robinson Luis S. PhishKiller: Uma Ferramenta para Detecç ao e Mitigaç ao de Ataques de Phishing Através de Técnicas de Deep Learning. In: BRAZILIAN Symposium on Information and Computational Systems Security. [S.l.: s.n.], 2019.

THAKUR, Kutub; PATHAN, Al-Sakib Khan. **Cybersecurity Fundamentals: A Real-World Perspective**. [S.l.]: CRC Press, 2020.

YASIN, Affan; FATIMA, Rubia; LIU, Lin; YASIN, Awaid; WANG, Jianmin. Contemplating social engineering studies and attack scenarios: A review study. **Security and Privacy**, Wiley Online Library, v. 2, n. 4, e73, 2019.

ZWILLING, Moti; KLIEN, Galit; LESJAK, Dušan; WIECHETEK, Łukasz; CETIN, Fatih; BASIM, Hamdullah Nejat. Cyber security awareness, knowledge and behavior: A comparative study. **Journal of Computer Information Systems**, Taylor & Francis, p. 1–16, 2020.