Sandboxed navigation and deep inspection of suspicious links reported by Humans as a Security Sensor (HaaSS)

Master of Science in Technology Thesis
University of Turku
Department of Computing
Cyber Security (EIT Digital Master School)
June 2022
Andrea Zanotto

Supervisors:
Enrico Frumento (Cefriel, Milan, Italy)
Petri Sainio (University of Turku)
Seppo Virtanen (University of Turku)

UNIVERSITY OF TURKU
Department of Computing

Andrea Zanotto: Sandboxed navigation and deep inspection of suspicious links reported by Humans as a Security Sensor (HaaSS)

Master of Science in Technology Thesis, 62 p.
June 2022

This thesis is part of a long-lasting research carried out in the field of Humans as a Security Sensor. In this thesis, I propose a solution to help companies to fight back against phishing, in particular, targeted and highly-contextualized attacks also known as "spare phishing". The thesis aims to develop a deep inspection module of individual emails submitted to the system by human sensors. As soon as a suspicious email has been flagged, it is passed to the deep inspection module that takes care of navigating every URL while collecting evidence and marks of malicious activities. The characteristic of this project is that it mimics the behavior of a real human user while navigating. It does not stop at the initial page, instead, it follows the redirects and collects page links to further inspect them afterward. My work focuses only on the automated navigation and deep inspection part and integrates it with an existing project that provides emails to analyze and manages the human sensor network. The idea is related to the concept of a human honeypot and provides a toolset that can help gather precious information to augment phishing user reports. We design a system that can navigate potentially malicious URLs as a human user would do. It opens links and browses through the webpages while collecting data, with the crucial difference that all the navigation is carried out fully automatically and in a protected environment isolated from the rest, so that any infection remains confined.

Keywords: phishing, spear-phishing, human-honeypot, HaaSS, social engineering

Contents

List of Figures

List of Tables

List of acronyms

AI	Artificial Intelligence
API	Application Programming Interface
ASE	Automated Social Engineering
GUI	Graphical User Interface
HaaSS	Humans as a Security Sensor
HSN	Human Sensor Network
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
IT	Information Technology
ORM	Object Relational Mapper
UI	User Interface
URL	Uniform Resource Locator *(the address of a web page)*
VM	Virtual Machine
VPN	Virtual Private Network

VPS	Virtual Private Server

1 Introduction

Phishing is a real threat, one of the most dangerous in today's cybersecurity scenario. Current approaches to prevent and mitigate the attacks work fairly well on general attacks but are not so effective against targeted ones like spear phishing. Spear phishing differentiates itself from general phishing because while the first uses a general message to induce people to give their information to the attacker, the second injects personal details about the receiver in order to make the message seem legit.

My work follows the lead of a previous project developed at Cefriel, which is a no-profit Italian digital innovation center founded by "Politecnico di Milano" and located in Milan. Cefriel is very active in the field of information technology, and it deals with digital innovation, research, and training. The existing project uses the human intuition at its foundations but augments it in order to provide a toolbox to securely analyze and scrutinize suspicious emails from the ground up. My work is focused on creating a deep inspection module for emails reported by human users and developing a secure and isolated environment where a human-like web navigation is performed fully automatically to explore potentially dangerous links and gather information.

This thesis aims to create the foundations of a framework to support secure deep inspection of emails reported as potential phishing. The project should be seen as additional protection built to catch threats that have eluded traditional checks. In particular, it provides an extensible set of tools focused on performing targeted deep inspections safely. For example, it allows URLs to be navigated deeply, which means that child pages and contents are also scheduled to be inspected together with their parent pages.

The thesis is structured as follows: Chapter 1 introduces and explains the rationale behind my work and what I want to achieve. Chapter 2 brings an overview of the phenomenon of phishing, the psychology behind its effectiveness, and the theoretical foundations upon which I based my work. Chapter 3 describes the design of the system, the decisions that have been taken, the ground on which I chose the components used, what was already done before, and what was planned in this thesis, as well as how this project combines with previous work. Chapter 4 exposes the implementation details, the tools, the components used, and how they communicate together. In Chapter 5, I explain what I achieved and the tests I did. Chapter 6 is reserved for discussion and future work, and then I spend some pages on the conclusion.

1.1 Phishing: threat and approach

The burden of keeping security in a company does not lay on the IT department alone. The importance of security must be felt by everyone, and it should be taken care of regardless of the department or the job carried out. Everyone needs to be responsible to put in place the necessary actions to avoid unwanted exposure to threats.

There are different approaches to empower employees and teach them how to recognize and react to immediate threats. One method is to use strong emotions to convey the message. Fear often serves the purpose because it is powerful and deeply rooted in our brain. In particular, creating policies bounded to fines, or establishing an environment of shame or punishment upon falling for a phishing campaign can be thought to be a good tactic to push people to be more vigilant. The problem is that fear lasts very short without a constant and active stimulus [1]. The result is that people (over)react in the moment, but then easily forget and move on, and very little is achieved in the long term.

To complicate things further, we are experiencing an evolution of attack tactics used to gain unauthorized access to companies' resources. In this thesis, I am analyzing social engineering focusing on phishing emails, and following this path, the evolution of the **semantic social engineering** has been identified as increasingly threatening.

Current solutions used to detect spam/phishing emails work decently against traditional attacks. Checking engines are currently able to compare a vast amount of signatures and apply heuristics to spot common details that are usually associated with this category of dangerous messages. These systems are widespread and largely used. For example, Google is able to filter out the majority of malicious emails well before they hit your inbox. Sadly, nowadays, phishing campaigns are moving toward **highly-contextualized** targeted attacks, and this is the area where automated solutions start to be much less reliable. The creators of phishing campaigns are aware of the filters that are usually used, therefore it is not so difficult to create a specifically crafted content that will silently sneak under the radar [2][3][4]. One possibility is to create a draft of the attack that is then submitted through a series of tests using common anti-phishing tools in the market. If the attack is being detected, the original message is slightly modified until it comes out clean from the tests [2]. Common changes consist of rewording the sentences with synonyms that do not trigger the heuristics engines, changing the HTML tags and attributes to modify the code without creating visual differences for a human, using unknown and clean domains, and many others. Once the engines do not recognize the email as a threat, it is ready to be deployed. Although these methods are routinely used in mass campaigns, there is also a more subtle and dangerous threat: **spear phishing**. This technique consists of creating a highly personalized text that is tailored to be credible and appealing to a specific target or a limited group. For example, it can be targeted to top managers of a specific company, the CEO, but also an interesting department. The campaign is much more effective the more information the attacker is able to gather and inject into the message. Knowing that the department has a vacation bonus can be used to create a fake website that asks for company credentials in order to obtain such bonus. This shift towards highly contextualized emails is challenging for automated (or AI-based) solutions that increasingly fail to recognize malicious content. The problem is moving from being a technical challenge with well-known markers (bad translations, strange domains, etc.) toward being an intellectual challenge. Since phishing emails are getting closer and closer to legitimate real ones (as represented in Figure 2.1), the only way to differentiate them is using intuition and, more importantly, an enormous amount of contextual details.

Although humans are naturally good at extracting context and linking various information, it is still a challenge for automated solutions. In fact, it is currently impossible to map all the relevant context information into a computer system. In the previous example about the vacancy bonus, some important context details would be: *"Is this website quoted in some official article or communication?"*, *"Were there webinars talking about this portal?"*, *"Did the company tell the employees that such a system was put in place?"*, and *"Was it stated that company-credentials should be used to access it?"*. All these questions can be answered quite easily by a human but they represent a serious challenge for a computer. Conversely, humans can be easily overloaded, stressed, or busy to the point they choose to blindly trust the claims of the email to avoid the burden of checking for inconsistencies.

It should be clear that we are outlining a scenario where both humans and computers have strengths and weaknesses. On one hand, automation makes the processing of large amount of data possible, on the other, humans are much more context-aware. People can make use of implicit information and intuition that are impossible to map on a computer system, but they can only process a limited amount of information before getting tired, losing focus, making errors, and trying to circumvent the system to ease the fatigue.

Cefriel is exploring solutions that can mix together prevention solutions with awareness. The direction that currently drives interest is the concept of **HSN (Human Sensor Network)**. The idea is to use the targets of phishing attacks as weapons to fight back by creating a network where every human can offer their cognitive insights and their natural ability to spot security violations to prevent others from falling victim to a phishing attack. This is based on the **HaaSS (Humans as a Security Sensor)** and **citizen science** and aims to combine the collective effort of every single individual to the creation of a secure and protected environment.

Citizen science is the study of the collective effort of a population (in our case, company employees) to collaborate and gather information and insight of a phenomenon. In principle, people are practically like live sensors that process information, distill it, and feed the result to a central system. The central system is, in turn, responsible for making informed decisions and possibly applying mitigations. In our scenario, humans measure the suspiciousness of the emails they normally receive and flag the ones that are considered a threat to the company systems. Examples are: possible viruses, invites to surf doubtful links, or direct attempts to steal credentials with phishing.

Every good-quality sensor must be tested and calibrated, and human sensors are no exception. Some people are naturally skilled at spotting something odd, some others do not have the expertise and the experience needed to achieve a decent detection. The more the sensor is tuned, the higher the standard of the reports.

The quality of the reports is a crucial trait: if we are in presence of noisy advisories, we need to waste a lot of resources to clean them, discard the errors, and crosscheck every claim. We are also slower to respond to threats because a user report is considered inaccurate by default and potentially wrong. Therefore, by the time the system verifies the claim, an incident might have already happened.

The target is creating an automated system that is called to act just after an email is reported as potential phishing by a user. The email is analyzed and all the web links contained in it are scheduled for navigation. An automated worker pulls the queued URLs and performs a human-like navigation, simulating the behavior of a real user while gathering as much information as possible. Simulating a real user allows the system to analyze links as if an employee had fallen victim to the attack, moreover, the software can navigate further the webpage by emulating clicks to UI elements in order to move from site to site searching for the attack landing-page where the employee credentials would have been stolen.

With this analysis, it is possible to confirm or deny the users' intuition using a deep automatic check without exposing them to any threat. The deep inspection is performed in an isolated environment and the user will only receive the final evaluation.

1.2 Motivation for the work

Cefriel has long been studying methods to decrease the impact of phishing attacks. One area that seems promising is using humans as a sensor to detect attack attempts [5]. However, humans are far from being flawless and unbiased, and there are numerous ways to trick them, as described in Section 2.2. Blindly trusting a report from a user and applying mitigation based on that can have disastrous consequences. For example, if the system decides to block a domain as a result of a user report, we might accidentally create a denial of service. Maybe the domain was not malicious at all, it might have been wrongly reported because it acts in a suspicious manner but it is ultimately safe to visit. Blocking it for the entire company would result in lots of people not being able to proceed with their work, but also a number of requests to the IT staff in order to revert the automatic decision. The base approach is to use human operators to confirm or deny the claims of the user reports. Intuitively, though, this only shifts the problem of the intrinsic human unreliability. We can train a small department to be highly qualified, but this does not scale well and involves great costs. The approach explored in previous Cefriel researches [5] was to statically analyze the content of the email, searching for common signatures and evidence of phishing. Unfortunately, in well-crafted content, this produces little result because attackers avoid well-known templates and common messages in favor of credible domains and links. Sophisticated attackers often create seemingly legit and good-looking walls which only serve as a gateway to lead the user to the real place where the phishing happens. With this research, the aim is to be able to recognize this pattern before the user becomes a victim. This is done by developing a more proactive machine that is not limited to analyzing static content, but actively searches for suspicious marks by following the path the user would have taken. We aim at pushing the door wide open to be able to smash through the wall of redirect-links or credible domains to reach the real stage where the attack happens.

Phishing is also lacking the presence of software that is specifically designed to deal with these attack schematics compared to other threats such as computer viruses, keyloggers, or ransomware. So, I think that I can give a real contribution to the state of the art by exploring this potential solution.

In addition, this method also tries to fix the human innate tension to curiosity. Many people feel the urge to know what there is behind the link of an email, even if they know it is most probably something malicious. The fact that an anonymous link is locking them out from the secret that is held behind is often unbearable. I myself felt that feeling and it is strangely hard to resist. Rationally speaking, there is no point in opening it, but still it is annoying not being able to. In a research done by Michael Stevens [6] they found out that “We dislike being bored so much that sometimes physical pain is preferable”. With my work, I provide relief to this urge because the link can be analyzed safely and the result can be seen securely afterward. For instance, a step of the data collection can include creating screenshots of the page behind the link that can be archived and seen without exposing the viewer to any threats. It is also easy to create a graph that quickly shows the hierarchy of the pages in a visual way. It can show the links inside, and the possible paths that every page can lead to. Everyone, being them an employee, a researcher, or a security worker, can access the data and understand the insight of an attack in a protected way.

The main objective that is being pursued is to create an automated service that can lighten the work of human operators, creating a system that can perform deep checks efficiently. Reducing the time between the attack and the deployment of the mitigation is key [5] because the sooner the attack page is blocked at company-level, the lesser people can fall victim to it.

1.3 Objectives of the thesis

1. Create a **web navigation system (crawler)** that emulates the human navigation while recording distinctive features that could be used to determine to what degree the website is a legitimate one.

   The system must be as **modular** as possible, I tried to create it in such a way that it is easy to change the engine that performs the navigation to meet specific needs. There are different types of web navigators that can be used to simulate, to various degrees, a human navigation. Each one has its peculiarity, for example, its speed (pages per minute), the elements that can support, (text, images), the actions that can be performed, and the number of details that can be collected (can execute JavaScript, can follow redirects, can take screenshots, etc.). While providing sensible defaults that work best in the general case, I decided to keep the implementation modular to allow quick and easy changes.

2. Create a **reproducible and automatically deployable sandboxed environment** that supports an isolated and secure way to navigate potentially malicious websites.

   As previously said, the navigation is extremely exposed to real-world threats. For this reason, it must be easy to deploy new instances of the workers or replace infected ones. I decided to use virtual machines for the isolation, but particular care has been put on using the best way to provide fast deployment and replacement of the machines. The aim is having a single click solution that takes care of creating a new instance of the navigation worker, downloading all the resources needed (including operating system, software, libraries, and third-party tools), configuring them, and connecting to the master server that dispatches the work to do. In this way, providing that the master server is up and running, the system can scale automatically based on the demand of computation and analysis. Moreover, individual instances can be replaced with a simple click: one virtual machine is destroyed and another one is initialized and provisioned to replace it. The system could also be programmed to heal itself by having the virtual machine to initiate an auto-destruction if some system file is tampered with, or after a certain amount of time. In its place, a new clean instance will be started.

3. Create a **deep inspection system** that provides a unified way to perform the analysis and store the results and the scores that could be then used to indicate the likelihood of a website to be a phishing webpage.

   All the crawled pages are extracted for as many details as possible. These are saved in a database where another component can pull the relevant data, analyze it, and make a decision on whether or not the webpage is malicious. The analyzers can be completely decoupled from the crawler and they operate independently. Moreover, modularity was a big concern during the design phase. The approach used consists of having all the analyzers independent of each other, and having the possibility to deploy them on different machines. When a new type of analysis is needed, developers only need to create the relevant module and make it connect to the main database.

4. Create the **infrastructure** that is responsible to store, retrieve, and analyze the data received by human reports as well as the deep-inspection score to create predictions.

   This thesis is not only concerned about trying to create a proof of concept, the objective is trying to elaborate a framework, a common base point that can be extended at will. With a strong infrastructure, extending it or changing the components only requires writing the glue code necessary to communicate with the central system that is then responsible to integrate and communicate with the rest.

2 Background

2.1 Phishing

Although the project can be extended freely and target different types of email-based attacks, I decided to put more effort into contrasting phishing. This decision is guided by the fact that phishing is one of the major threats in terms of cyber-security for a company. The FBI's Internet Crime Complaint Center, in its 2020 annual report, states how the number of incidents related to phishing has consistently outnumbered all the other types of cybercrimes in the last five years [7]. The trend is nowhere near getting better, actually, the very contrary is true. Almost all cybercrimes are growing in the average number of events, and a big gap can be seen between 2019 and 2020 where phishing events more than doubled, closely followed that others that have also seen a big growth.

Among the causes that led to this spike in cyberattacks, there is for sure the tragedy of the COVID-19 pandemic. As Steven Merrill (FBI Financial Crimes Section Chief) said: *"Unfortunately, criminals are very opportunistic. They see a vulnerable population out there that they can prey upon."* [7]. This state of emergency is having as a consequence the massive digitization of almost all work sectors. This was a necessity to ensure safer working conditions, so working from home has become the normality for many people, and the urge for reducing close-contacts moved many in-person actions to be replaced by online forms and web applications. The issue here is that this innovation was *forced* to happen, and often companies were not ready to educate their users about this abrupt change.

Scams can happen in every context, but in-person ones are usually more difficult to design and carry on. A criminal cannot fake a bank building, fill it with fake workers and wait for people to get inside. The example is, of course, an exaggeration, but it serves the purpose. Creating an online scam is much easier and leaves fewer traces behind. The nature of the internet itself is an advantage for the attacker because while people are more used to remembering a bank's physical address or the faces in the office, they are much less aware of the digital address of the same bank and the details that distinguish it from a clone. The impersonality of the internet leaves people with fewer resources to recognize a threat. In today's world, it is easy to predict a growing trend toward digitalization in the following years, and if no countermeasures are applied, we will probably witness at least a proportional growth in incident events and losses.

Phishing is also lacking software that can proactively protect users, especially when the attack is targeted, because the most used approaches today are based on deny-lists or simple heuristics. This is partially due to the intrinsic difficulties to instruct a computer to process and understand context, a cognitive process that we, humans, possess innately. The other reason that prevents the use of such elaborate systems on a large scale is that they require complex reasoning and information gathering, therefore they necessitate too many resources to be run on massive inputs. Deny lists or simple heuristics can scale very well, which is good, especially when considering big email providers that have to scan a huge amount of traffic, but, as we said, they provide limited capabilities.

2.1.1 What is phishing?

Phishing is the art of convincing someone to spontaneously offer their personal information to the attacker without even realizing what is going on and the threat they are in. This is usually done by mimicking someone, or something, that the victim trusts: an institution, an organization, or a friend. The purpose is being able to gather the victim's personal information, which can be of any kind. Common examples are images, names, passwords, and PINs, but the list is not complete, in fact, anything that allows the attacker to later impersonate the victim is interesting, and the more information the attacker gets, the better they can build their subsequent attack. The discovery of personal information is indeed not the real target, it is often just a transitory stage. In the final phase of a phishing campaign, the attacker exploits the information or the access gained to perform the real attack. The targets can be varied and range from gaining access to bank accounts to perform wire transfers, accessing other online services to buy items without consent or using the benefits the user has in them, installing malware, or stealing important information. Usually, the most valuable information that users can give to criminals is the one that allows them to escalate the attack and obtain further resources or higher-level access. The most straightforward example is user credentials. They are easy to fill inside a crafted webpage and the user is not so alarmed because inserting credentials is a daily activity required for many common tasks.

To create a credible trap, attackers often copy the user interface of the subject they are impersonating. The visual imitation is usually powerful enough to mislead because it is the part that covers the majority of the screen, and a familiar interface puts people at ease [8]. Visual similarity is important for both the bait email and the landing website. Today's websites usually rely on a clean and simple UI, both for aesthetics and for user simplicity. On the other hand, this simplicity is advantageous for the attackers because they can replicate the contents easily. Manually recreating the interface provides the best results, especially in terms of detection avoidance, since a clean fake just created does not present a known signature. However, a quicker way is also available. There are a huge number of ready-made templates for the majority of the most famous websites that can be customized and used right away [9][10]. The URL is another important part of the visual clues. Many awareness campaigns underline how important the **correctness of the URL** is [11], but the problem is that recognizing a malicious URL is far more difficult than it seems. First of all, it is not trivial to remember the exact domain of every company. There is no standard or widely accepted best practice. Every company picks the name that fits their standards the most, therefore every combination of words and special characters *could* be valid. Criminals are well aware of this human weakness and often use crafted URLs that contain the name of the targeted service along with other words that differentiate them from the real website. Examples are the use of subdomains like `www.famousbank.website.com`, in this case, the main domain is `website.com` which is completely unrelated to famousbank. Clues about the targeted website can also be put after the domain, like in `website.com/famousbank`. Similarly, the main domain is again `website.com`. GlobalSign suggests avoiding links with unnecessary words or text around the domain [11], but the suggestion here opens to a lot of interpretations and errors. There are many legitimate examples where words around the domain can be considered suspicious but are legitimate instead. A trivial example is `https://sustainability.google/`: here *google* is the top-level domain, which is very uncommon for a company. But also `https://experiments.withgoogle.com/` is a legitimate Google website, even if the word *"with"* could be considered unnecessary. Phishing websites are also often disguised behind **URL shorteners** [12][13]. Different online services offer short anonymous URLs that redirect the user toward the real destination. When this scheme is used, it is difficult for the user to know the final landing page without using external services.

Of course, a shortened link is not automatically a sign of malicious content. The original purpose of a link-shortener service is indeed to create a more manageable URL that users can copy manually, not hiding the real web destination, and it is therefore used for many legitimate purposes. A more subtle way to trick the users are homographic attacks which exploit the similarity between letters in different alphabets. To give a real example, the websites `https://www.paypal.com/` and `https://www.paypal.com/` are visually identical, but in the latter, the second *a* is in reality a Cyrillic letter. One can check this by putting it in the search bar of a browser, the result will be the following punycode[1] representation of the URL: `https://www.xn-pypal-4ve.com/`.

2.1.2 Cost and profit of a phishing attack

Criminal activities do not differ much from regular business if we look at them from a strictly economical point of view. In fact, both involve many risks, very different in kind and impact, but similar at the concept level. To run a phishing campaign there is the need to invest money, time, and resources to generate revenues. Moreover, criminal organizations rarely, if ever, run solely on their own strengths. The complexity and the level of expertise needed to manage every aspect of an attack are simply unmanageable. There are too many services, tools, and infrastructure needed to be profitable without the help of external parties, especially when the attacker wants to scale to big numbers quickly. Criminals rely on various suppliers during the attack stages, which, in turn, may or may not be legal. The services required vary hugely, also in relation to the attack, but often attackers need hosting, channels to distribute malicious content, account checkers, proxies, VPNs, and other anonymization tools. However, often, cybersecurity is much more expensive

[1]Punycode is an encoding procedure that allows the Unicode charset to be represented using the limited ASCII subset. In this way, any special character, emoji, or characters from other alphabets can be consistently represented.

than cyberattacks [14]. There is a huge and unfair discrepancy between the budget allocated by companies to defend their assets and the cost barrier for the attackers. In a 2018 report, [15] Kaspersky found that enterprise companies are spending on average 9 million dollars for cybersecurity, which corresponds to 26% of the entire IT budget. For small and medium businesses the situation is pretty similar with the average spending of 23% of the IT budget, corresponding to 246 thousand dollars. And the trend is nowhere near to decline.

Cyberattacks are easy to set up and criminals can start monetizing very rapidly from them. The cost barrier is minimal for a basic attack, even if it creates a big impact. The internet is a limitless source of tools and knowledge, and it is not difficult to find free tools to perform cyberattacks. For example, there is a range of free tools aimed to perform penetration tests and security evaluations that can be turned into a weapon, they have a legitimate purpose but can be used to create harm as well. Weaponized software is also available in the black market and can cost as little as a coffee. Fake webpages for phishing or simple password cracking tools can be found for 2$ as well as Wi-Fi cracking tools, keyloggers, or Bluetooth hacking tools for roughly the same price. Malware and Trojans can be more expensive, averaging around 50$, but the low price is not an index of low quality. Many of them have great potential to disrupt the activity of a business and create further damage. One can easily find leaked FBI/NSA Hacking Tools for about 5$ even if the value of that technology is worth thousands of dollars. The fact that the software can be easily replicated and sold makes it virtually cheap despite its power. The only notable exception is the trade of zero-day exploits that can cost as little as 3000$ [16][14]. The dark web, however, does not only offer technical tools, but also knowledge and step-by-step guides to create a powerful attack from scratch for just 10$. Therefore, not only the entry barrier for criminals is very low, but also the preexisting technical skills needed can be very limited or nonexistent. Even a novice can quickly be guided through the various stages of the attack and learn the basics with a good return of money.

That being said, buying malware and reading a guide is not all is needed to launch any serious attack. Sure, it can create disruption in a limited scenario, but in order to launch a serious attack that can scale and can generate a sustainable amount of profit, the attacker needs an infrastructure that can support the creation, delivery, monetization, and anonymization of the attack. For example, running a campaign to distribute a banking Trojan typically needs the support of at least 5 or 6 different services [17]. For that reason, the price of the individual tools is often not enough. Deloitte has studied exactly the total final cost from the attackers' point of view, which allows us to calculate the return on investment and the profitability of the attacks. The study shows that a million-dollar-impact cyberattack can be set up with as little as 35$ a month, and could return a net gain of 25000$. But the returns can be much higher, even a one million return with an investment of about 4000$ a month. The most inexpensive way to reach those numbers seems to be phishing kits, according to the study [17].

To state some examples, a complete solution for phishing is one of the easiest to set up. The basic components are the hosting (which can be obtained by renting a virtual server or VPS), the phishing kit composed of a fake webpage that resembles the official one, and a distribution method which is often targeted/bulk emails. The average cost for all the services is around 500$ but can range between 30$ and 1600$.

For now, the difference between the low entry price of an attack and the potential for high profit is simply too high. One way to stop this trend is to introduce sophisticated defense measures and continuous innovation that forces cybercriminals to constantly adapt and change their arsenal to be effective. Reinventing the attacks and creating always new tools would be an added cost for criminals in terms of time, money, and effort. This shift has the potential to eventually change the adversary's cost-benefit scenario and therefore reduce the criminal catchment area.

2.1.3 How big is the impact of a phishing incident?

On top of the expenses and the assets put in place to mitigate and prevent attacks, companies must also face the consequences of successful attacks. In 2018, the average cost of a data breach was more than a million dollars, with 120 thousand on average for small and medium enterprises [15].

According to the last IBM report, losing a record of personally identifiable information costs on average 180$ to the attacked company [18]. This includes various metrics and activities. For example, we must take into account the fines that may arise in case of an incident. Some countries have very strict rules when talking about user data and privacy, and the violation of the policies might lead to high fines. An example is GDPR. Then, there is the cost related to the incident response, including people working to solve the cause and the consequences of the incident, to return to a safe environment. Calculating the total impact of an attack is often very difficult because we cannot only take into consideration the money spent, but also the future consequences, even if these factors are often intangible, variable, and difficult to quantify. In terms of dollars and cents, a data breach can cost over 4 million dollars on average [18][17], but the long-term impact can cost hundreds of millions if not billions. In fact, to assess the long-term consequences we have to consider the concept of *lost business* as an important factor. Losing confidentiality over intellectual property, contracts, or other confidential documents can be a disaster for a company to the point it is no longer able to continue its operations sustainably. A business might be forced to temporarily stop its operation until the attack is contained, which not only stops the production (and therefore the revenues), but can also lead to fines if the company agreed on a binding date for product delivery to clients. In addition, important incidents resonate very quickly with the public, which translates into reduced sales and deals due to the decreased trust. The credit rating can also increase, leading to costlier loans.

2.1.4 Phishing types and vectors

There are multiple vectors to deliver a phishing attack, but the most common is through emails. There is usually an initial distinction based on the user base toward which the campaign is directed. **Bulk campaigns** are often general and low quality but exploit the law of large numbers. In other words, they attempt with so many people that at least someone is likely to fall victim to it. To quickly gain user trust, banks or famous online companies are impersonated by the attacker, but it is also common to receive messages that are allegedly from well-established authorities such as the police or the government. The more well-known the impersonated entity is, the more the chances that the user has used their services. These are the campaigns that are easier to recognize and block because the large majority of people mark them as SPAM or phishing, and the content, the links, and the sender are quickly put into public lists and automatically blocked from that moment on. If we consider, instead, more targeted emails aimed at a specific group of people or a company, we talk about **spear phishing**. In this attack, quality is more important than quantity, the emails are tailored to the target. The attacker may also take time to understand the habits of the target(s), the company policy, near events, and use them to include critical details that make the attempt credible and ease the flow for the victim. While in bulk-phishing the attempt is the general gain of money, power, information, or goods; in spear phishing, the aim is usually much more specific. The targets are usually high-profile figures that have access to critical company resources and information or employees in the financial sector [19]. The attack is usually aimed and has a specific mission to accomplish.

The historical behavior of these kinds of campaigns is very distinctive. As we can see in Figure 2.1, spam emails are very broad. There is little to no context or personal details so the delivered message could apply to virtually anyone opening the email. This ensures paying little effort in order to reach the maximum amount of people. Real emails contain precise context and details, there is no need to be vague or broad, in fact, the opposite is true. Spear phishing tries to mimic as close as possible that level of accuracy and details, but remains still distinguishable from a legit message by a trained expert. What is scary is that the probable future trend will be a more contextualized content that will shrink the gap between an attack and real emails to the point that it will be hard, if not impossible, to discriminate.

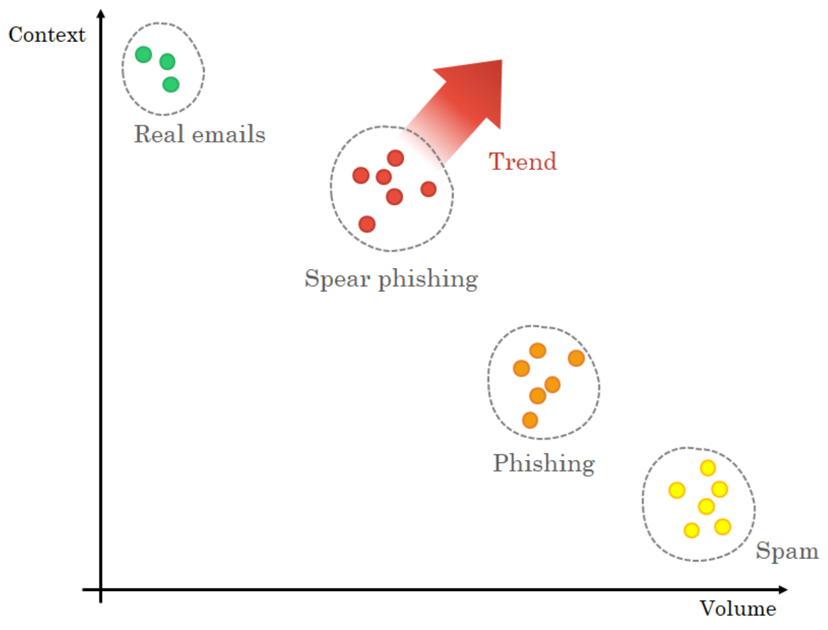

Figure 2.1: Spear phishing trend

The increase in volume and a more precise context inside phishing emails are caused by various factors. The evolution of security systems and the constant awareness campaigns are pushing criminals to enhance their techniques, study the target, and gain information. Every year there is new software to help reconnaissance and there are great tools to organize the knowledge gained. For example, Maltego [20] allows for creating detailed views of the data but also increases productivity by using multiple external data sources and collaborating with others. However, the giant leap in terms of increasing the number of addressable people is led by the rise of artificial intelligence and automated social engineering (ASE) [21]. Even with today's capabilities, an automated botnet is able to alter and shape an online environment without the direct intervention of a human [22][23][24]. People can be hooked by using chatbots that realistically imitate a human conversation. This can be especially effective when receiving fake support emails. The user can be attracted to the chatbot that behaves like a human technician that needs the user data to proceed with giving help. This is much more effective and less suspicious than a simple form where the user puts its data. And in the future, human-to-human conversations can be entirely automated, removing the human attacker from the equation and therefore allowing to scale the attacks. Google Duplex has already proved to the world that complex phone conversations can be handled by a computer without the interlocutor noticing, [25], and even if Google's tests were focused only on taking appointments and reservations, we are not very far from a general-purpose solution.

2.2 Social engineering

All the tricks and behaviors that the attacker exploits for convincing the victim leverage a series of techniques that falls under the umbrella name of **Social Engineering**. In an attempt to define it, we can borrow what Hadnagy wrote in "The art of human hacking": *"the act of manipulating a person to take an action that may, or may not, be in the target's best interest"* [26]. Of course, social engineering can be used in a variety of contexts, but when applied it leads the subject to an accommodating mental state.

As Joseph M. Hatfield says in "Social engineering in cybersecurity: the evolution of a concept" [21], *"[the] education and training of potential victims is the best way to prevent [phishing] attacks within the cyber domain"*. For social engineering to work, a fundamental assumption is that there must be an unbalance between the victim and the perpetrator. In this case, we have a discrepancy of knowledge between parties. The idea behind this is also referred to as epistemic asymmetry. One party has a high degree of knowledge on a subject and uses that to instill a change in the behavior of others that are in a position of decreased power.

2.2.1 The psychology of phishing

Humans, the weak link (?)

"The weakest link in computer security sits between the monitor and the chair".

The literature is crowded with quotes similar to this one. As mentioned in Section 2.2, there are multiple ways to hijack the human brain and push a victim toward a preset goal. People seem to be extremely vulnerable to biases, surprisingly brittle in making estimations, and prone to be deceived. Just to state an example, in an episode of Mind Field, [27] an experiment shows how the majority of the participants can effortlessly provide a fulfilling reason for having chosen a photo that they actually had rejected. They were, therefore, providing a reason for a choice that they never made, or more specifically, that they made *in reverse*. People are also much more concerned about a threat when they are not in control of the situation and have to trust others [28]. And again, experiments like the Asian disease problem [29] show that people react in completely different ways when the same problem is stated as a gain or a loss. They come to opposite solutions even if the problem is ultimately the same, just stated in a different light. The bias, in this case, is that people tend to overestimate the gain and neglect the losses. Similar reasoning can be applied to phishing, where the users tend to overestimate the favorable opportunity (as a promised bonus or a prize) over a potential danger (the possibility to fall victim to an attack). Another innate bias that is often exploited is the tendency to overreact to stimuli that are spectacular, rare, immediate, and unfamiliar. On the contrary, stimuli that have been internalized over a long period of time, that are common and familiar are perceived as less important, regardless of the real danger they carry [28]. Cyberattacks start as an “unfamiliar” danger, for example after people are made aware of a problem such as phishing. They tend to overreact and put in place strong security measures that, in turn, reduce usability. After some time, the threat slowly fades into the “familiar” category and the strict measures are felt like a superfluous burden and are likely removed or ignored. On top of that, there is the confirmation bias: our decisions are biased toward what we already believe, and it is therefore difficult to exit the tunnel of misconceptions and biases. To make things worse, we must stress that even if people are generally persuaded to have good detection abilities in terms of phishing, some studies show that in reality they perform poorly at tasks aimed to test their ability to spot deception and lies [30][31][32].

Understanding the human brain

In order to have a better grasp on the underlying rationale behind human actions, it is important to understand and study the processing unit of our decisions: the brain. Evolution always pushes to maximize the chances of surviving and proliferating, and since our brain is a product of evolution, it is no exception. The most primitive system that is in charge of evaluating risk and reacting to it is the amygdala, an almond-shaped area of the brain close to the center. Among its functions, it is in charge of processing sensory inputs and emotions and putting together a response in the quickest time possible. For example, it is involved in immediate decisions that must be taken when lacking the time for a more profound and conscious evaluation. This might seem like a lifesaver, which indeed it is, but it has been trained for a long time to be suitable against predators and life or death situations. However, the world we live in today is far more subtle and complex than that. Almost all of the threats we encounter in life are not white-or-black, they are better represented as shades of possibilities governed by complex factors such as the probability that a threat will occur, the possible impact it may cause, and the mitigations that we can put in place to reduce its impact or the likelihood of occurrence [33].

The amygdala is, however, only a part of the picture. In fact, in order to better predict the changes in our environment and make accurate forecasts, mammals developed a different area of the brain, the neocortex, which specialized in accounting for complex high-level reasoning and analytical thoughts. The neocortex is therefore exactly what it is required to ponder over probabilities, impact, and mitigations. Nevertheless, higher processing power comes at a cost: a much narrower pool of information that can be processed per second [33].

The fascinating result is that two completely different systems are called on to judge the same dangers and to provide countermeasures to the same threats simultaneously. However, while the amygdala is fast, effortless, and operates at the subconscious level, the neocortex is slow, requires cognitive effort, and can only process input serially. They take part independently in the decision until the reconciliation of the results, but given the neocortex drawbacks, it is often challenging to overcome an amygdala's decision. This is problematic because albeit the amygdala is blazing fast, it is often influenced by emotions, biases, and past experiences, and it is hard to consciously override its decisions. On the contrary, the neocortex can be willfully modified and evaluated, and it is also flexible enough to fit new environments and challenges. It uses rationality, which is what is needed to process probabilities and outcomes that are described in the previous paragraphs, but it falls short in competing in speed and cheapness. And to complicate things further, not even the neocortex is immune to errors as it tries to cut corners with stereotypes and rules of thumb to reduce the cognitive burden.

We seem to be programmed to be bad at reasoning

What seems to emerge from the analysis is that humans seem to be almost *programmed* to be bad at reasoning, with our intrinsic biases and laziness. But looking at a different perspective, what if the flaws were intentional? Something we do not use reasoning on is intuition. We do not have to reason in order to understand if someone is happy or upset, we just intuitively know. We do not know exactly what brought us to that conclusion. Sure, we can reason on the clues that led to the intuition, but it is a follow-up process, not the one that initiated the decision [34]. On the other hand, if we do not have an intuition right away, or it turns out to be wrong, then we can sit down and use rationality, and even when doing so, we experience deep biases and mistakes; although they might be there for a reason.

Hugo Mercier and Dan Sperber in their book [35] hypothesized that reasoning has not evolved for us to make better decisions, but to be a better social animal. Our strength as a species is our extraordinary ability to cooperate and help each other to achieve a common goal. For example, when people are asked to choose from two identical products except for the fact that one has more functions, they tend to choose the one with more features even when they do not plan to use them or do not like them. The grounds for this decision might be the fact that it would be easier to justify to *others* why they pick that product. Or, to say it with different words, they can avoid being embarrassed to be asked why they choose the one with fewer features [36].

Unfortunately, while this works when cooperating, it can become harmful when someone exploits this mechanism for its interests. The attacker breaks this environment of reciprocal cooperation and turns it against the victim.

2.3 Citizen science

Citizen Science is the idea around the collaboration and knowledge-sharing between scientists and a (usually large) group of enthusiasts and curious people who want to help research. People that are willing to join their forces toward a common goal. [37][38] This group does not need to be composed by scientists. Quite the contrary is true, in fact, a community can even be composed of a heterogeneous group of people not necessarily belonging to an established scientific community. Private citizens, enthusiasts, and hobbyists can all participate together by providing data, checking, analyzing, and finding patterns. A lot of activities do not need a lot of training and are easy to complete, but they may require an immense amount of manpower which is simply infeasible for any research team. Those are perfect candidates for citizen scientists to be involved. People can volunteer and donate time and resources for the research. An example can be counting the number of species of birds that can be seen or heard in a specific area. For example, the North American Breeding Bird Survey does just that since 1966 [39]. Another example can be manually labeling photos of plants or animals, or picking up samples of botanical species in a precise area.

Huge datasets are vital for research purposes: the more the samples, the more information can be extracted from them and the hypothesis generalized. This is true for the natural world as for the cyberspace. Virus samples or signatures of phishing emails can be a great advantage against attacks, but they can also be analyzed to spot new trends and to study how the threats are changing over time.

The main principles the citizen science is based on are not at all new, in fact, some hints trace the collective effort of recording information for the greater good of the collectivity even in ancient China [40].

These principles are very simple [41][40]:

1. **Anyone can participate.**

There is no limit to the people that can take part in the research, and this is its major strength. It is based on the community and this gives a potentially unbounded amount of resources. In addition, these resources can act independently of each other, and they can work in parallel, the only condition is that they need to coordinate with a common system to gather the data.

2. **All the participants use the same protocol to exchange information.**

   One of the most important aspects to consider when gathering a huge amount of data is being able to use them afterward. This seems like an obvious requirement, but it is often misunderstood. For example, having a huge amount of unstructured data is quite the same as not having it at all. The process of cleaning the data often requires the same amount of effort as recording it, and it usually must be done by a human. Of course, cleaning the data after the recording does not scale. For this reason, all the participants must use common protocols that ensure interoperability and an easier subsequent analysis.

3. **Data gathered help to reach an objective.**

   Data must always be relevant to the main objective of the study and they must be of good quality. And of course, the objective must be a real scientific question that generates new knowledge or understanding.

The principles of the citizen science are already used in several security-related projects, even if it is not explicitly mentioned. Antivirus vendors are using this approach since long ago [42][43][44]. The users are the perfect sensors because they routinely find new unseen files. It would be impossible for a single company to examine all the possible threats in the wild, even more, promptly respond to something that has just appeared. On the contrary, if the user opens an unseen file, a sample can be sent to the antivirus vendor for analysis. In this way, the analysis is carried out just after the exposition, minimizing the time needed for a response.

Citizen science is linked to the concept of the wisdom of the crowd: a large enough collection of people can process information better than any of the individuals alone. In fact, when it comes to big decisions, we often prefer to have a number of people arguing and providing reasons for their claims rather than letting one person decide, even when this person is knowledgeable. Having a plurality of views increase the probability of opposite or conflicting ideas, and there are more chances that the individuals' biases are smoothed out among the crowd [36]. Citizen science goes in a similar direction: the system can tolerate mistakes from the participants because the results are "averaged".[2]

2.3.1 Humans, the mitigation

Humans are often pictured as "the weak link" or "the problem", but I am convinced that while this might be true, it instills a sense of misery in people that can eventually lead to thinking something as: "people are intrinsically condemned to fail on security, so there is no point in putting effort into it". At Cefriel, we think that people can be instead a big part of the solution if they are empowered in the right way.

Given that attacks are specifically aimed at human weakness, and that automatic solutions fail to distinguish them, a good counterattack is, in fact, humans. Why? Because they have the cognitive abilities to extract the context and reason about it. This is not the only motivation, though. Performing any kind of deep inspection requires a huge amount of time and resources and does not scale with large volumes as a result. Performing deep analysis is therefore infeasible if applied to all the emails that reach the company email server. On the other hand, humans automatically and effortlessly perform deep inspection at a subconscious level when looking at the email subject. They often perceive a gut feeling about the email, but they might lack the tools and knowledge to verify it, and they may decide to continue

[2]As long as the quality of the data gathered is good.

interacting with the content for lack of alternatives. If we mix human intuition with resource-intensive deep inspection, we have the right weapon. The human trigger ensures that the resources are not used blindly, and the limited workload allows more in-depth analysis.

Moreover, having a collection of people that give their opinion allows for reducing the relative presence of false positives and false negatives because we can even out people's individual biases and errors and use the collective knowledge of the crowd.

2.4 Human Sensor Network HSN & Humans as a Security Sensor HaSS

2.4.1 Weakness of automated solutions

There are automated countermeasures that try to fight phishing, this section tries to give a high-level overview while underling the limitations and the weaknesses of the current solutions.

One big class of phishing countermeasures falls into the category of anomaly detection. The information is scrutinized and searched for odd details that are strange enough that they might be a signal of phishing. This includes looking for unusual characters in the URL, long links, redirections, but also domains concealed behind IP addresses or obfuscated characters through encoding [45]. After the analysis, the email is assigned a suspiciousness score and, in case it exceeds a threshold value, it is considered phishing. The technique offers good results, but it is really hard to place the threshold: too low and the system will be flooded with false positives, too high and it misses a lot of threats. The problem is that there is not an absolute good value for the threshold. Sometimes legit websites use suspicious URL schemes but are ultimately safe to visit. `https://experiments.withgoogle.com/`

seems odd, we would rather expect `https://experiments.google.com/` without the extra *"with"*. However, it is a valid Google domain and not a phishing attempt.

PhishTank [46] uses a different approach: users can submit phishing websites so that the whole community can use its APIs to check if a suspicious link has already been reported. It is basically a giant blacklist that anyone can navigate and use. Many companies use PhishTank and other blacklists to protect the users, but these kinds of protections are only effective when someone else has already reported the link, which is extremely unlikely for company-targeted phishing.

Machine learning is often used to spot phishing emails. The implementation can vary, for example, it is possible to create a model from datasets of known past phishing emails and pose the question as a classification problem. The input data can be sourced from the suspected URL or the text of the email. [45] However, this generalizes well only when the new emails share common traits with the ones that were identified in the past. Spare phishing, however, plays at a different level. It does not try to persuade the user using classical techniques that are meant to be effective toward the general crowd. Instead, it creates highly personalized content that mimics as close as possible legit conversations, therefore these emails are invisible to the eyes of the machine as it lacks the underlining context to discriminate.

On top of that, there are tools [3][4] that allow testing an email body toward common signatures tagged as phishing before sending. In this way, it is trivial to use synonyms or introduce unimportant differences that elude the controls. This process can also be automated by letting an artificial intelligence come up with the perfect combination of terms that minimizes the suspiciousness score.

2.4.2 The power of human insights

Targeted phishing aims at sneaking out of two detections: automated detection run by machines, and human detection. They are able to fool the machines by creating highly contextualized messages that are identical to real ones from the machine's point of view. But not for humans. We people have the cognitive tools to be able to discern the difference, get critical insights, and make use of them. What is hard for humans is detecting typos, subtle but important differences in how the URL is written, or since when a domain has been active. They are all details that are straightforward to spot for a machine. For that reason, I am convinced that, with the help of machines to augment the data, humans can be a decisive tool to spot phishing and protect an entire company from phishing threats.

Of course, human sensors need to be weighted. This ensures that we are reacting proportionally to the potential threat. Previous experiments at Cefriel [5] using simulated phishing campaigns suggest that every sensor should be evaluated similarly to how we monitor hardware sensors. The detection rate measures the portion of the emails recognized as phishing over the number of phishing emails they are exposed to. The accuracy is the portion of real phishing over the number of emails reported as such. These two indicators are useful as a proxy to evaluate the human sensor. But another statistic is crucial to understand the effectiveness of the human sensor network: the response time of an individual. This measures the time that passes between the reception of a phishing email and its reporting. This time is influenced by many factors, often not related to the effectiveness of the single sensor. For example, if the email is received outside the working hours there are fewer chances that it is read, and therefore reported. The same applies to different hours around the clock, scheduled meetings, and so on. But even if there is great variability and it is often unreliable when taken individually, it can be a good indicator of the entire HSN when averaged.

The response time can be thought of as a way to sense the herd immunity of the company to phishing. The reason is simple: as soon as an email is evaluated as phishing, all the dangerous contents (links and attachments) are obscured company-wide, blocking every infection from that moment on. So, even if a few employees recognize the attempt, it is blocked for all other people on the target list. The sooner an email gets reported, the sooner the mitigation can be put in place and more credentials can be saved from the attack. A company with a very low response time can even tolerate a consistent number of cyber-clumsy employees because they are passively protected by good reporters.

There is strong evidence [47][48] that if you ask a crowd to estimate a physical quantity and you average the guesses, the result is surprisingly accurate. In two famous examples, the number of jellybeans in a jar or the weight of a bull has been tried multiple times, and, consistently, the averaged number has been far more accurate than any individual guesses [49]. However, not even the crowd is always right. An important constraint that has been shown to improve the score significantly is that every guess should be independent of each other. If that is not the case, the final result tends to drift towards a misplaced bias [47]. Another thing that we can do to improve the reliability of the final guess is to take expertise into account. In fact, we can use a weighted average with a parameter that measures how good a specific sensor is at getting the right value [50].

During the initial evaluations, the human sensors not only provide feedback to evaluate the incoming emails, but also give the system the possibility to estimate their level of expertise in spotting attacks. Knowing which sensor is more vulnerable to attacks does not just play an important role in the evaluation, but it can also be used to protect the company even further. We can imagine a system where people who consistently fail at noticing phishing emails gain additional protection. In particular, we might selectively delay the reception of untrustworthy emails for those vulnerable individuals, so that by the time the email reaches their inboxes, it has likely already been diagnosed by the rest of the network.

2.4.3 Report augmentation

The result of the human report gives tremendous insights on possible threats, and it could also be used alone to deploy countermeasures such as blocking a URL company-wise or disallowing the download and execution of an attachment. However, we can go even further by collecting information and crosschecking user claims thanks to data augmentation. Reports are a powerful weapon, however, sometimes they might still be ambiguous and inconclusive, there might be not enough reports to justify a block, or maybe we want to dig deeper and try to find hard evidence without exposing anyone to danger. After all, human reports as based on gut feelings, in fact, we do not want users to open potentially dangerous URLs to double-check. The system should take care of this process securely, explore potential threats as if it was a user, and report back all the evidence found in a clean and structured way such that a security expert can review it.

In addition, all the generated data are not reserved for experts, every user can see the analysis of the email they reported. In this way, we can provide feedback on the user's gut feeling about the email, and we can satisfy their curiosity by giving them a glimpse of what the email URLs were hiding.

3 Design phase

3.1 Project overview

This thesis is a complementary work and a continuation of long-lasting research at Cefriel that aims at lowering the impact of phishing attempts that employ highly contextualized targeted attacks. In particular, this thesis aims at extending the excellent work of Fabio Menzaghi "A Human Sensor Network approach against phishing attacks" [5] performed at Cefriel. Menzaghi's thesis is focused on providing a framework that employees can make use of to easily report phishing emails when they hit their inbox. The process outlined in his study describes a system that tries to automatically categorize reported emails based on their content, including embedded links and attachments. After the categorization is final, a feedback analysis is sent to the reporting users so that they are informed about the result of their intuition. Moreover, the feedback is also a moment of teaching, because it includes an overview of the deceiving elements, even the ones that the user might have missed. The end result is to spot real phishing emails and flag them so that others are prevented to interact with them, or at least strongly advised against. The system is aided by a mix of automated analysis on the email text with the supervision of a human operator that certifies or corrects the automatic prediction.

My contribution is similar but complementary. I envision a system that takes the reported emails and performs a deeper inspection of the contents by performing a human-like web navigation of the links in the email. During the navigation, the system collects all the links and gathers information about the context and the behavior of the websites. All these details are stored and associated with the URL they belong to and may then be used to decide whether the website is malicious or not. In addition, since we are navigating on potentially dangerous websites, we need to take measures to prevent any possibility of malware exfiltration into internal systems. The navigation system is the most exposed because it has to physically interact with malicious contents. If the attackers are able to find and exploit a vulnerability at any point of the stack used, the system can be compromised. This puts in danger both the information gathered that could be tampered with or deleted, but can also poison the decision process, making it (partially) ineffective. To build a ring of protection, exposed components are enclosed inside a sandbox that isolates the environment from the host computer. For maximum protection, a virtual machine hosts the navigation system which communicates with the rest of the software via a network socket.

All the analysis starts with a user report, the reporting system could also be extended to take into account various heuristics to consider the performance of the human sensors and their reliability. Then, the automatic system performs the analysis. This automatic step is important for various reasons. First of all, it reduces the amount of work required by operators that manually check the reports. In the near future, there will be still the need for manual high-level expertise to sort the most tricky kind of phishing emails, but with automation it is possible to sort the easiest ones. This ensures that the operator is only asked to handle difficult cases, avoiding the boredom of trivial ones. Additionally, the details gathered alleviate the effort of repetitive research for the operator that can just review the gathered information and focus their effort on more complex (and probably more stimulating) activities. To sum up, the system can help the security team in various ways: it lifts the heavy part of gathering details for the decision, it automates most decisions, and it retains the data gathered in a structured and indexed way for historical purposes.

3.2 Secure sandboxed environment

In order to create a secure environment, I started to scrutinize current isolation technologies, as I had little previous knowledge on this topic. At first, I considered containerization techniques such as Docker [51], but I then tried to broaden the research to find the best solution. During my research, I was concerned about the possible achievable security while keeping the infrastructure relatively easy to manage and configure. Using a complex solution can lead to difficulties down the road, especially when the system needs to be maintained and upgraded. When the complexity of the system becomes substantial, it is likely to introduce security issues from a misconfiguration, and since the infrastructure I am designing is in its very early stages, there are not enough resources to manage a complex environment while guaranteeing safety. Moreover, I am trying to build an isolated environment that can be activated automatically, therefore predictability is of the utmost importance.

Containers, such as Docker-based ones, are often referred as to lightweight virtual machines that offer the possibility to isolate a portion of the system without having to virtualize hardware and not even an operating system. Containers run on the same kernel of the host machine, but there are restrictions put in place to create a barrier between the rest of the system. Containers use `chroot` to create a new filesystem root so that the host files are kept secure, while it is possible to share host files and folders by mounting them. Resources isolation such as the sharing of CPU, RAM, and group of processes is managed by `cgroups` which is a part of the kernel subsystem [52]. The security of containers is really high, however, they are referred to as a partial-virtualization solution [53]. They are usually characterized by a mixture of privilege restrictions, virtual filesystems, process confinement, and network isolation; nevertheless, they are ultimately running in the same kernel of the machine that is hosting them. All their actions are mediated through the kernel but, in case of a vulnerability in it, the processes inside the container might escape and gain high privileges inside the host machine [54][53].

As Jens Erat points out [55], Docker and other similar container solutions use a "bottom-up" approach. You start with having the application running with the same kernel, and then you raise barriers around it using kernel namespaces, cgroups, and others tools. This offers better performance, but at the same time, it is difficult to be sure not to have left anything behind because the surface at stake is very large. Docker itself states that *"One primary risk with running Docker containers is that the default set of capabilities and mounts given to a container may provide incomplete isolation, either independently, or when used in combination with kernel vulnerabilities."* [56].

On the other hand, full virtualization such as virtual machines uses a "top-down" approach. They start with very good isolation by default, and only provide *some* well-guarded and well-described interfaces to control and access things outside the virtual machine scope. In this configuration, the virtual machine uses a private kernel, different from the host's one. If a kernel exploit is found and exploited, a malicious process might escalate privileges, but the hypervisor is still guarding the access to the host machine. Of course, it is possible to find hypervisor issues that can be exploited, but the attack surface is greatly reduced, and even when such vulnerabilities are found, they are usually very complicated and limited in scope [54][55].

I moved on to review sandbox software in order to analyze common techniques used by state-of-the-art solutions and try to understand what are their strengths and the pain points. For example, Sandboxie [57] is a sandbox-based isolation software for Windows operating systems. However, by using it, we would be limited to one operating system and there are still problems in isolating some programs [57]. In addition, Sandboxie is still a partial-virtualization solution and from my analysis, there are not enough mitigations that differentiate it from a containerization solution like Docker in terms of security.

Since my research suggests that good isolation can be achieved with the use of a virtual machine, I focused on that field. However, one of the requirements is to create a one-click solution that can download all the necessary resources and prepare the isolated environment without human intervention. Going through the steps of manually preparing and provisioning a virtual machine was not acceptable, not even counting the time required for such a process. Creating virtual machines should be done with ease, therefore, the research shifted to finding a manager that takes care of building, maintaining, and deploying virtual environments. As I looked for solutions, Vagrant by Hashicorp [58] started to emerge. It is a tool that allows the desired state of the virtual machine to be specified using a text file. Vagrant takes care of downloading the base image for the VM and provisioning it by specifying the steps required to download the components and software required. With a single command (`vagrant up`) a new instance is created with the same characteristics as every other clone of it. Moreover, Vagrant manages all the lifecycle of the VM, including its destruction. By default, it uses VirtualBox as a backend (called provider), but it is not limited to it, for example, VMware and Hyper-V are supported, and others can be used with a custom configuration. In the end, I am satisfied with the isolation provided by VirtualBox, while the ease of controlling every aspect of the VM lifecycle with Vagrant made it a perfect solution for the project.

Using full virtualization penalizes the performance, but I consider that as out of scope for this thesis because we lack the resources for exploring efficient methods and modifying existing solutions to increase execution speed. My tentative is to create a viable solution with the maximum level of security I can achieve given my knowledge and resources. Other considerations can be found in Section 6 as future work.

3.3 Web Navigation

One of the main components of this research is the human-like automatic web navigation. To manage the communication with the various browsers, I planned to use a middleware to ease the development and to have a common interface to control various browsers. The first candidate was Selenium [59] which is a stable and long-lived software for web automation. It uses WebDriver as the underlying protocol to communicate with the browsers [60], which is the component that translates the high-level operations to commands that the browser can execute. Using Selenium, I was able to create a basic but functional navigator, the problem, however, lies in the limitations of WebDriver. Its current implementation does not permit bidirectional communication[1] which limits the amount of information that we can extract from the browser. For example, by using Selenium (and therefore WebDriver) the network operations performed by the browser are completely transparent to us. We cannot detect a redirection because we can only indicate the page to visit and wait for the end result. All the intermediate steps are not exposed through the APIs.

I moved to a tool that is able to create a bidirectional communication with the browser and access a lot more information, which is valuable in the context of gathering details that can then be used for the analysis. Playwright [62] is a very recent piece of software in active development that offers this extended set of capabilities [60] and I decided to use it as the default for the project, while giving the possibility to change it for Selenium or scrapy in the configuration. Being a new entry in the field, Playwright has some minor drawbacks, for example, it is guaranteed to

[1]Although a new protocol called WebDriver BiDi might allow bidirectional communication in the future. Currently, its state is a draft [61].

work only on very recent versions of the browsers and the use of Firefox currently requires a modified version of it, but the development team assured that this is likely to be improved in the future [62].

For test purposes, I also let the navigator engine be changed to scrapy which is a fast web scraper [63]. It runs orders of magnitude faster than Selenium or Playwright, but it lacks a lot of capabilities used to gather vital information and can only process static webpages because it does not process JavaScript content.

3.4 Deep inspection

During the navigation, we are interested in gathering key information about the page. The focus is to create a module that can be easily extended to capture the information that is deemed relevant. Instead of focusing on deciding whether or not the page is phishing, I put together and store the information that can determine that, but the ultimate decision is left to another project, likely aided by an AI.

The navigation is the first place where data can be collected. During the navigation, the software recognizes and stores all the HTTP redirects that occur before landing on the final page. In this way, it is possible to record the steps between a link created with URL-shortener services and the real destination. Moreover, it collects all the links it finds inside the page, both to spot if the page is the gateway to something malicious, but also to navigate them further if needed. With this implementation, we have a large degree of interaction with the web browser. Playwright gives complete access to the page, so we can download the complete HTML code, make screenshots, but it is also possible to access the cookies in the session, the localStorage content, the HTTP headers, the IP of the server, or the status code of the response. Basically, everything accessible to the browser can be captured. What should be stored, instead, depends on the subsequent analysis that will be performed on the data. The project is meant to provide sensible defaults but can be extended at will depending on the area of focus, the space available, and the computational resources usable.

As soon as a new page is discovered by the crawler, it is possible to perform additional forensic analysis. The system provides convenient hooks that can be used to send the information acquired to external services that can perform deeper data manipulation and analysis. To provide an example of the functioning, I wrote an integration that sends all the links found to `urlscan.io` via API calls that respond with a series of data, including a suspiciousness score calculated by their servers. Of course, this is only an example, information can be sent to any external parties, or it can be submitted to an internal machine in the same network, or to a software program running on the same machine. The possibilities are endless, the idea is to provide a seamless framework that integrates all the data manipulation services together.

4 Implementation

For the implementation, there were very few constraints. The main one was that the core part of the project have to be developed using `python`. It is not a technical limitation but rather a decision made by Cefriel. Moreover, since the project is experimenting with a new concept, I tried to make the system as flexible as possible in order to keep future changes manageable. Using proprietary solutions in the early stages is likely to influence the future path of the project, and it can also limit the freedom to experiment due to vendor lock-in[1]. Every component is designed to be replaceable so that later optimizations and changes can be done without having to start from scratch.

Ubuntu is used as the operating system for the virtual machines, while other components (such as the database) are run inside Docker containers.

4.1 Infrastructure - components

In the design, the first distinction is between long-lived components and ephemeral ones. The central database is an example of a permanent component, it stores all the historical knowledge gained, and it remains stable throughout the system lifetime. Its content can be updated and moved, but the component itself remains static. On the other hand, ephemeral components have a more dynamic lifecycle and

[1]Vendor lock-in occurs when a user becomes dependent on a specific vendor since changing it to a new one would lead to substantial switching costs.

are generally considered expendable and designed to be easily and quickly replaced. An example of those is the worker nodes, for instance, the nodes that perform the automated navigation, or the ones that retrieve additional information and analyze the data gathered. The intelligence usually resides in the long-lived components, instead, the workers play a rather passive role in the system. They are assigned a task by the central database and they execute it. The prioritization of the tasks and the logic reside in the permanent nodes. Workers only need to complete the task and report back the result. If, at any time, the task execution gets stuck or there is a security problem, the worker node is decommissioned and possibly replaced with a new working version. Workers nodes are designed to be stateless, in this way the system can scale according to the load and the tasks can be distributed freely. In addition, their stateless nature ensures that they can all be considered equal, and when a failure is affecting a node, the work can be redirected to another worker without the need for special care.

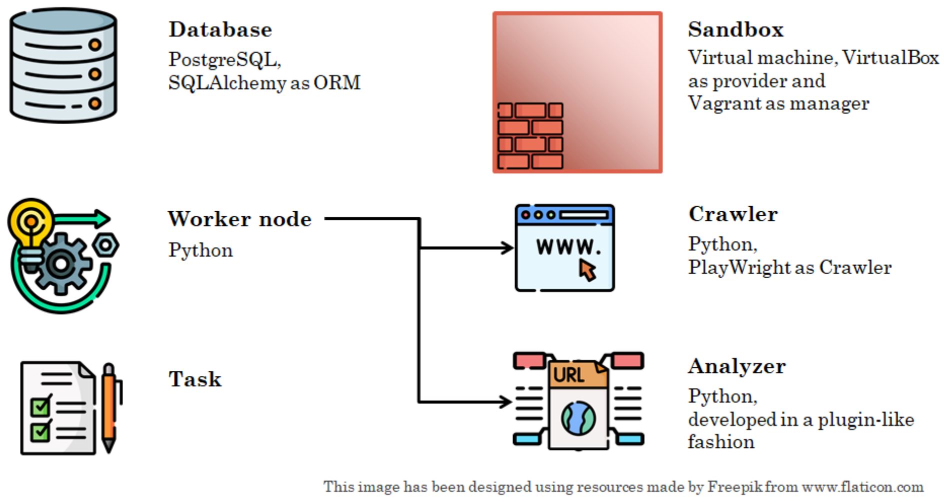
Database
PostgreSQL,
SQLAlchemy as ORM
Sandbox
Virtual machine, VirtualBox as provider and Vagrant as manager
Worker node
Python
WWW.
Crawler
Python,
PlayWright as Crawler
Task
URL
Analyzer
Python,
developed in a plugin-like fashion
This image has been designed using resources made by Freepik from www.flaticon.com

Figure 4.1: Infrastructure components

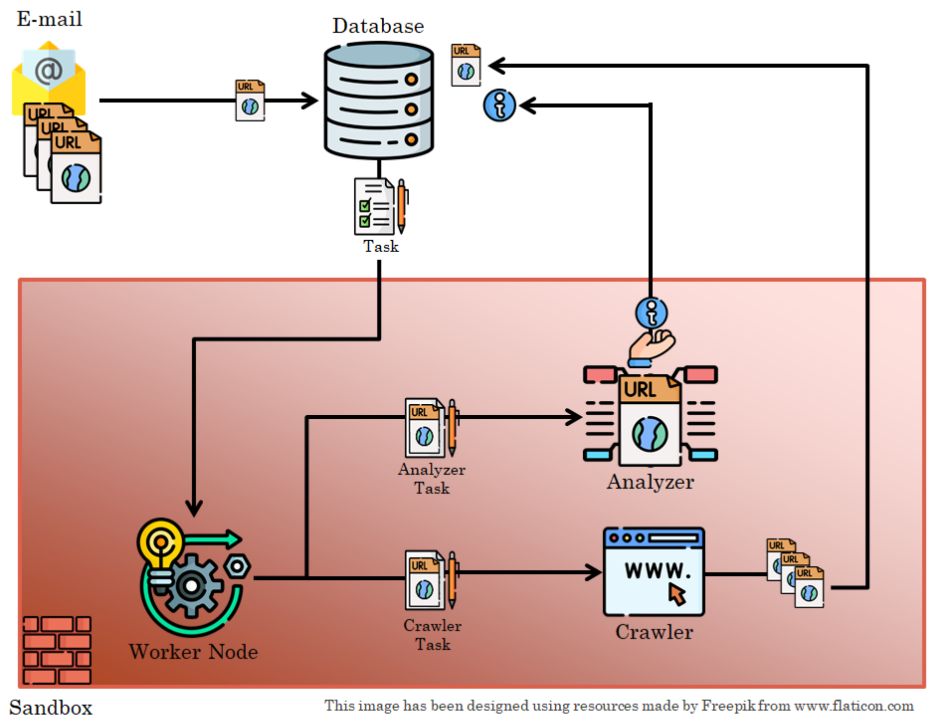

Figure 4.2: Interaction between components

4.1.1 Database

The core element in the infrastructure is the central database which serves as the main information container. It stores all the information gathered and also the entire state of the system. It gives a glimpse of all the tasks that are being performed by all the other components and allows tracing back past executions and failed tasks that need to be recovered. With this approach, all other components can be considered ephemeral and expendable. The system can tolerate failures on the ephemeral components as long as the central database remains consistent. In case of a worker failure, its tasks will remain pending, but they will eventually be detected, recycled, and scheduled to another worker. Since worker nodes are considered insecure by default, they only store information that is strictly needed for the current task execution. When the task is completed, all the information is moved to the central database, and the related task is marked as done. Before receiving other instructions, all the temporary data is cleared and the system is initialized for the new execution. Keeping the data away from the workers has multiple advantages. First of all, it needs to be protected in a single place, it is therefore easier to create policies and enforce them. Moreover, keeping the worker nodes lightweight means that they can be replicated faster, in fact, no state replication nor synchronization is needed. We just need to deploy the code for the task execution, the rest will be dynamically provided.

The database can be deployed freely and only needs to expose a port to make the data accessible. I decided to deploy it inside a Docker container to ease the development, but the production database can be deployed in any environment. It can also be extended to work with a distributed database, as long as it exposes the same interface.

PostgreSQL [64] runs the database, however, the code interacts with it through a layer of abstraction by using an ORM. The employed library is SQLAlchemy [65] which is a stable and open-source ORM solution for python that gives a seamless interface to interact with different relational database systems. It simplifies the code by giving convenient methods to read and write from the database, and it also creates a standard interface to interact with different database systems even when they "speak" different code dialects. Changing the database is therefore manageable and only limited changes need to be performed in the code.

In this prototype, worker nodes have direct access to the database and they can query it freely. However, this is only a convenient shortcut for development and should be restricted in production. For example, the database can be protected by an API layer so that a worker can only access the information related to the currently assigned task. In this way, we prevent attacks and accidental deletion.

4.1.2 Tasks

The minimum amount of work that can be done in the system is defined by a **task**. Every task contains the directives for the execution of the work, its constraints, its rules, and the expected result format. Creating a task is an immediate action, the caller does not wait for its execution, therefore, creating a new task is the standard way to schedule any kind of job in the infrastructure. It is a promise of execution. In this way, the caller can be stateless. There are a number of task types, one for every action that can be executed. The main task type is the crawler one. It stores the URL to visit along with details such as the depth of the navigation and the link position on the page. These details are used by the crawler to decide whether the navigation is needed or it is going "too far" or "too deep". The task has also a container for the results, therefore the execution is always bounded to the results it has achieved. In the crawler example, a result can be the redirections found during the visit, the links inside the webpage, but it can be extended to contain also screenshots, the entire HTML code, HTTP return codes, and all the relevant details needed for the subsequent analysis. The task format is strictly dependent on the service it supports, for example, when querying an external service, typical result fields are the ones returned by the service APIs. Every task also stores its current status. The main statuses can be: `READY` when the task has just been created and it is ready to be picked up, `WAITING` when it is assigned to a worker but waiting for execution, then `EXECUTING`, and finally `DONE` or `FAILED`. The task format is flexible and tailored to the information needed, the only constraints are that every task must be bounded to the URL it refers to, has a valid state, and a recognized type.

4.1.3 Sanbox

The approach chosen to fulfill the sandbox requirement is to use virtual machines. Although they require much more resources compared to other solutions, and there is a non-negligible spin-up time, they offer the most security. In the end, I use Vagrant for managing the virtual machines in a local environment, but this is only a convenience for development because the nodes can be easily deployed inside a cloud datacenter.

I did not spend much time optimizing the performance of the machines, since I considered that out of scope for the current development stage. The operating system of the sandbox machines is a standard Ubuntu 20.04 with long-term support. I use a pre-made Vagrant image, and everything needed is installed through provisioning scripts. The provision stage can be long, but it is possible to create a custom image equipped with all the tools and the code necessary and use that as the main image. By doing so, the provisioning is done only once, which speeds up the process but also ensures that every copy of the machine is identical to others.

Vagrant gives the possibility to specify every configuration of the virtual machine using a text file. Manually executing the steps to deploy a virtual machine would be a huge burden for an operator, even more if we need to deploy multiple virtual machines and we have to check and recreate some of them. The overhead would be unmanageable, not even counting the likelihood of human errors. A text configuration, on the other hand, is easy to maintain, can be put under version control, and can also be used to trace back problems.

First, I changed the default Vagrant configuration in order to give the machine a fair amount of resources, the GUI mode should be enabled so that the browsers can be executed and the user can visually check the navigation. Since browsers are mainly created to be used by humans, they expect to find a display in the machine, and failing to provide one can lead to errors in the execution. I decided to enable the GUI mode of VirtualBox which creates a window that shows the virtual machine display output. This intrinsically creates a display on which the browsers can render the pages without errors. I also adopted this approach so that we can keep an eye on the browser execution. However, the GUI mode is not strictly required. I successfully tested the creation of a virtual display so that the application can be executed on hosts that do not provide a physical display, usually servers or a cloud environment. Then, I also disabled some default behaviors used in development, such as the folder shared with the host machine, and I only share the project files in read-only mode. As the last step, I install and configure all the dependencies with provisioning scripts, and I automatically create a python environment in which the code is executed. All the configuration and provisioning are automatically performed, then, after the machine boots, the python program is executed and starts to execute the tasks that it receives.

4.1.4 Worker nodes

Worker nodes are composed of a python core that takes care of all the business logic. When the node boots, it executes the main python program. The first thing to do is to connect to the database. We can configure the node and specify what types of tasks it will download and execute. Doing so, allows us to prepare the nodes with a task-specific configuration. For example, nodes that crawl data should be equipped with browsers and a good amount of RAM and bandwidth, a node that performs machine learning might benefit from good graphical cards, instead, minor data analysis can be run on inexpensive nodes, or in background. By default, every node receives every type of task, but I encourage the use of task-specific nodes for the best performance.

Nodes can download from the database a configurable number of tasks in a single call, to reduce overhead. Every node can also read a configuration file with parameters that affect the execution. For crawlers, we can configure the number of concurrent pages navigated, or the browser to use. As soon as the task is ready, we try to navigate the URL in it. Using Playwright, we can detect any redirection occurring, and store it in the database. Once we land on the final page, the crawler extracts all the links in it and creates new tasks to navigate those. At every step, the depth of the navigation is updated in the new tasks. In this way, we can prune the research tree when we go too far. A similar approach is taken for the number of links on a page. It is possible to set a limit so that only the first `N` links are visited. This pruning is necessary, otherwise the research space would grow exponentially and it is not guaranteed to terminate. I defined some sensible default values, but it is also possible to override them per task. If we want to investigate further a specific domain or link, we can increase the allowed depth, or reevaluate them with more extended parameters in case of inconclusive evidence.

Other workers use a similar approach, the task APIs and the interaction are the same, what changes are the parameters stored inside the task and the fields that accommodate the results. For an analysis, common fields might be the suspiciousness level of a URL, and other fields that explain the reason for that decision. Other analyses might include meta results, for example aggregated statistics: the topmost used word on the page, or the number of occurrences of a specific term. Other tasks can elaborate results from the meta-analysis and take decisions upon them.

4.2 Flow of information

The system presented in this thesis should not be considered an alternative to current anti-phishing solutions. It is more of a safety net when all the previous ones fail.

Figure 4.3 explains it clearly: if all previous filters fail and a phishing email reaches the inbox of company employees, it can be flagged by a human as suspicious. Previous work at Cefriel [5] takes care of evaluating the reliability of human sensors and creates a first guess on whether or not an email contains phishing. My system is a complementary tool to this because it can automatically analyze the URLs and gives better insights to decide whether the content is malicious or can be considered

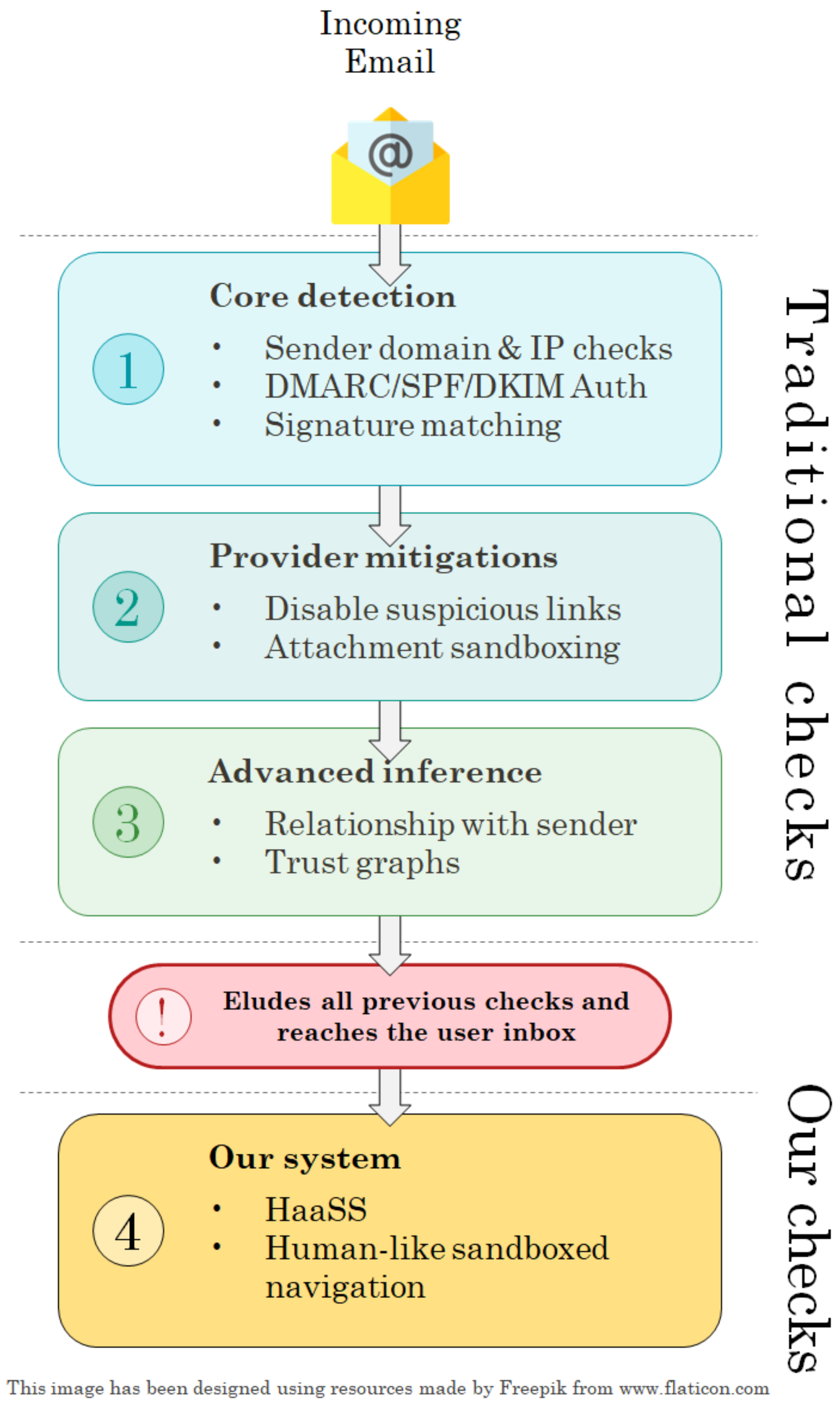

Figure 4.3: Integration between this project and existing anti-phishing tools

safe to visit. Moreover, all the analyses are performed in a protected environment so that no one is ever exposed to threats in the process of determining the nature of the content.

As soon as the email gets flagged, we can create a task for its analysis[2]. The task is inserted in the database in `READY` state, where it waits its turn to be delivered to a worker. The first task is always a crawler task because we need to get information before analyzing it. The system, however, does not blindly execute the tasks given, instead, it uses a memoization-programming approach. Before executing the task, it checks if it has already been completed before. In case it is, the result of that execution is used instead of triggering a re-execution. A clear example can be done with the crawler. Let us have a website `A` that has in it the URLs `B` and `C`, and let us say that we already visited all of them. If, during a research, we find a website `Z` which contains a link to `A`, we do not need to navigate it because we already know its content and the content of its children (`B` and `C`). In this way, we use past executions as a cache, which allows us to speed up consistently the process. There are, however, ways to invalidate the cache, for example, if we consider the snapshot too old and worth redoing. The cache mechanism is also smart and allows us to visit children that were skipped due to being too deep if we encounter them shallower in the navigation tree.

While we collect the navigation tree rooted in the reported email, we can start to submit the completed pages to the various analyzers that digest the content found and gather additional information. Similarly, only new unseen tasks are executed, while others use cached contents. Every task type can have an independent cache-invalidation algorithm that fits with the nature of the analyzer. Analyzers that digest static content never invalidate the cache, others that depend on variable and external factors must choose a conservative value that ensures the validity of the data with a good probability. For example, the ownership of a domain name is an

[2]The behavior can be configured, of course. We might wait for more people to report the email before starting the analysis, to reduce the resource consumption, but I think that a prompt reaction ensures better results.

example of a property that might change as time pass, even if the domain remains the same.

5 Testing and Evaluation

For the testing phase, I use a mid-range PC running Windows 10 64bit. The CPU is an AMD Ryzer 5 2500U with 4 physical cores and a total RAM of 8 GB. The guest virtualized machine is run by VirtualBox with virtualization enabled (VT-x/AMD-V). The VM has access to all 4 cores and a dedicated 4 GB of RAM. To keep the tests manageable I always analyze a maximum of 10 links per page, and I navigate to a maximum depth of 5 levels (excluding the initial page)[1].

For the tests, I first started using legit websites in order to check the ability to navigate correctly on harmless websites. The first tests are carried out starting from a simple website of a local company[2]. Note that even if the tests are repeated with the same starting URL, every run has slightly more or fewer total URLs. This is caused by the dynamic nature of some websites that load different contents when opened multiple times. For example, advertisement contents are loaded randomly and can contain different links each time. Similarly, random slideshows or other recommended links are often loaded randomly and can change the outcome of the overall navigation.

Here are the results of a run using the VM and 4 parallel crawlers.

[1]The initial page is considered level 0, its children level 1, and so on.

[2]I used a local company because it usually mimics well possible phishing. Using big company websites such as Google or Amazon produces an enormous amount of URLs, too many to run reasonable tests.

#	**Websites visited**	**Total URLs found**	**Aborted (out of scope)**	**Cache hit**	**Total time**
1	94	3610	156	3360	485 s
2	93	3590	159	3338	457 s
3	93	3590	158	3339	419 s
4	93	3590	160	3337	482 s
5	112	4003	170	3721	433 s

Table 5.1: Test in VM, 4 crawlers, Chrome

The amount of RAM used during the tests has always been constant at 2 GB, even if the available memory was 4 GB. During this test, the overall throughput was less than 5 seconds per crawled page. I am considering only the *crawled* pages because I noticed that they take orders of magnitude more than aborted/skipped URLs. This was expected because the cost associated with non-crawled URLs is only related to the database access, while for others we must interact with the browser, wait for input/output procedures, and render the contents. Another thing that is worth noticing is that common websites usually contain a lot of links that eventually lead to the originating page, creating loops. The system takes advantage of that by skipping the navigation in case it already opened the same URL. As can be seen in the Table 5.1, on a total of ∼3500 URLs about 95% of them were duplicates.

I also tried to raise the number of concurrent crawlers to 8, but the performance was significantly worse, as Table 5.2 shows.

#	**Websites visited**	**Total URLs found**	**Aborted (out of scope)**	**Cache hit**	**Total time**
1	93	3590	156	3341	546 s
2	106	3856	171	3579	721 s
3	93	3590	158	3339	647 s
4	154	6275	281	5840	957 s
5	105	3835	163	3567	863 s

Table 5.2: Test in VM, 8 crawlers, Chrome

I think that, using this setup, by increasing the concurrency we only add overhead and no performance boost. Therefore, the number of crawler threads should be decided based on the hardware used.

I also tested how the system behaves with no concurrency (Table 5.3), using only one crawler. The performance degraded a bit, as expected. There are many input/output operations when communicating with a browser, not even counting the time needed to open and render the page. For this reason, having concurrency allows us to fully take advantage of resources, avoiding wasting time waiting.

#	**Websites visited**	**Total URLs found**	**Aborted (out of scope)**	**Cache hit**	**Total time**
1	93	3590	157	3340	604 s
2	93	3590	159	3338	623 s
3	93	3590	158	3339	703 s
4	93	3590	159	3338	692 s
5	93	3590	157	3340	661 s

Table 5.3: Test in VM, 1 crawler, Chrome

From what I saw during the tests, using Firefox instead of Chrome produces negligible differences (Table 5.4) so I consider both good candidates for the navigation. Both browsers are mature enough and have good performance. I suggest the use of both browsers because some exploits might activate only in presence of a specific browser and remain hidden for others to elude security scans.

For reference, I repeated some tests in the host machine to see the performance gain. These tests are performed with the same program but without the protection

#	**Websites visited**	**Total URLs found**	**Aborted (out of scope)**	**Cache hit**	**Total time**
1	103	3647	170	3374	434 s
2	80	2790	132	2578	307 s
3	84	3106	156	2866	353 s
4	79	2768	135	2554	303 s
5	84	3009	139	2786	334 s

Table 5.4: Test in VM, 4 crawlers, Firefox

of the sandbox. The performance gain is about 100% (Table 5.5) which is huge, but it is quite expected because of the overhead introduced by VirtualBox. However, I suspect that other providers might have better performance. Moreover, I must stress that I am running VirtualBox over the host operating system, which is not ideal. More tests should be performed with the sandbox running in a cloud environment. I suspect that this change would improve the performance significantly, as the cloud environment is designed to run virtual machines with minimal performance penalties.

#	**Websites visited**	**Total URLs found**	**Aborted (out of scope)**	**Cache hit**	**Total time**
1	87	3439	153	3199	205 s
2	88	3460	150	3222	204 s
3	87	3439	144	3208	234 s
4	83	3227	144	3000	221 s
5	87	3439	151	3201	184 s

Table 5.5: Test in Host, 4 crawlers, Chrome

When the system navigates a URL, the result can be processed and then interpreted as a tree. For example, when visiting `https://bit.ly/3HAb4Ng` the result is the following.

```
[0] https://bit.ly/3HAb4Ng
 `--> [1] https://ark22give.com
        `--> [2] https://ark22give.com/#participate
        `--> [2] https://ark22give.com/#rules
        `--> [2] https://ark22give.com/#top
        `--> [2] https://ark22give.com/#transaction
```

In particular, I notice that behind the shortened URL we find a malicious website and all its subpages. Every page detected can be sent to various analyzers, both public blacklists and information gatherers. For example, we can access information on the domain and discover that it has been created two days before being found in an email in the wild. This is a mark of suspiciousness that will be taken into account during the evaluation. We can then analyze the text inside the pages and detect which brands are addressed. Then, we can calculate the types of URLs inside the page. In this case, there is no external link, and all the internal links point to the same landing page. This should also raise some suspicion.

6 Discussion and Future work

For the sandbox part, I am satisfied with the level of security that we were able to achieve, however, it can be extended even further by enhancing the general-purpose hypervisor of the virtual machine with a custom hypervisor built with forensic security in mind. Chris Greamo and Anup Ghosh [53] illustrate some examples of such a system that broadens the capability of the hypervisor by implementing integrity checks, data collection, and 0-day real-time detection. If performance is more important than security for the company use case, it may also be possible to loosen the isolation constraints and work in a more free environment. However, this must be evaluated carefully and adapted to the threat scenario and the company setting. In addition, I think that loosening the security must be paired with external systems that guarantee the containerization of escaping threats. A possible improvement to this solution is using other virtual machine managers. Vagrant is designed to be used in development environments, even if it can be modified and extended to be used in production. It serves the purpose, but as future work, I think it would be beneficial to analyze other implementations.

Regarding the automated navigation, I was able to obtain a good working prototype, however, the choice to use ready-made software to aid the navigation (Selenium, Playwright) has both pros and cons. They offer a simple and consistent interface to interact with major browsers, and they try to maintain compatibility and standard methods over time, which is valuable and is the reason I chose them; nevertheless, I must acknowledge that they are primarily designed to *test* websites. They are focused on an area that is not exactly what we are trying to achieve, although it has major overlapping with the concept of automated navigation. All in all, I hypothesize that better results could be achieved by directly using browser's remote protocols such as `Marionette` or the `Chrome DevTools Protocol (CDP)` because they would offer maximum flexibility and a wide range of actions.

There are a lot of corner cases when creating automated human-like navigation. This is particularly true if the objective is to mimic the navigation to the point that even when the attacker is trying to spot evidence of automatism or isolation, they cannot find any. Therefore, I think that more tests and real-world examples should be carried out to enhance this solution. Another area that should be analyzed better is the detection of JavaScript redirects. The detection can be achieved in different ways. A possibility is analyzing the JavaScript code, searching for hints of redirection and links. It can be hard to find every possible redirect, even more when the code is obfuscated, but a static analysis ensures the best performance. On the other hand, a more reactive approach is simpler to implement but has bad performance: we wait some time to allow the page to trigger a delayed redirect. This always works and does not need complicated scripts to analyze JavaScript code, however, we would pay the comfort with a much longer execution time. We must, in fact, wait on the page to see if something happens. This can be improved to only wait when we detect a timer-like countdown that is usually present in combination with timed redirects, but we are still wasting resources in a busy wait.

In addition, I think that additional tests should be carried out in a controlled environment. I did the tests using a virtual machine that runs in a host computer that can affect the result of the computation. Moreover, doing tests in a single machine needs a lot of time for a few tests because they need to be performed sequentially. I think that it would be better to test different instances of the program in a cloud environment that ensures more consistent conditions and allows us to run various tests concurrently, saving time without affecting the results. In this way, it is possible to find the perfect parameters that ensure good performance while minimizing the resources needed.

7 Conclusions

Phishing is a dangerous attack, even more in its targeted versions. The semantic gap between real emails and phishing ones is shrinking, leading to a more difficult detection by standard techniques. Companies need powerful tools to protect their employees, and they need a framework that makes the collaboration between users and the security team easy and effective. I tried to create the foundations to make this happen. I envision more secure email investigations and a generally quicker response to incidents. To do this, I created a module that automatically navigates the internet in a human-like fashion while gathering information. I designed and created this module in such a way that it can be confined in a secure environment that isolates it from the rest of the resources and contains any contingent infection. I also developed a way to automate the destruction and creation of the sandboxed environment, and I created a framework to connect a number of integrations that analyze the data gathered and collect more insights when needed.

The checks performed by this software should not be considered an alternative to current anti-phishing tools. It is rather complementary. In fact, in case an attack is so well-made that it can elude all the checks, we trust human intuition to spot the threat, and use the navigation system and deep inspection modules to confirm or dismiss it. It is like a safety net to catch users before they fall victim to criminals. The level of details that I am trying to achieve with this inspection requires time and resources to be done, it is therefore unfeasible to run over a large volume of emails. It should be considered a tool that performs a very detailed analysis when needed, in other words, every time there is a suspect. The human trigger ensures that no resource is wasted because the majority of emails are not scanned. For the remaining ones (classified as suspects) the system collects information and reports back the findings. Everything is done securely and with little human intervention so that the security team can focus on more important threats and leave tedious tasks to automation.

I am confident that the deployment of this automated solution as a countermeasure against phishing can be a game-changer and can help many companies to limit the impact of the attack or even intercept them before any harm is done. In 2021, an IBM report on data-breach costs says that companies using automated solutions and AI experienced a cost difference of nearly 80% compared to businesses that do not rely on these tools [18]. They associated this gap with the ability of automated tools to quickly detect the presence of a breach and deploy countermeasures. The automation helps to reduce the reaction time, which greatly limits the success of the attack. My objective is to offer a new tool that can help users and security teams to collaborate together against phishing. I try to offer a standard platform that can perform dangerous tasks in a controlled and isolated environment, lifting people from the burden of checking and deciding whether or not an email is secure or dangerous. Every time there is suspicion, the user has just to report the email and the system will automatically take care of collecting information and evaluate the threat. The generated information serves two purposes. If the data are conclusive, and we clearly detect malicious behavior, we can automatically confirm the report and take countermeasures. In case more research should be carried out, an expert can make use of the data collected to reach a conclusion. The security team does not need to waste time collecting information from (potentially many and different) services, they just have to review it. On top of that, the information is persistently stored and indexed so it can be easily reviewed in the future.

To sum up, I am satisfied with the results achieved and I think the objectives of the thesis have been met. The navigation system works to a good extent. It cannot yet perform complex navigation or peculiar edge cases, but it works well on basic cases. However, its implementation can be extended and the use of browser's remote protocols can help achieve a better outcome. The sandboxed environment is working well, including the automatic deployment and lifecycle management performed by Vagrant. The performance of the virtual machines can be improved as a future work but it is already good enough for reasonable analysis. The skeleton for the deep inspection is mature. I already implemented some modules that extract precious details about analyzed URLs and they can be extended easily. Moreover, new modules can be plugged in with minimal effort. Last but not least, the general infrastructure responsible for managing the entire project logic, such as the management of the tasks and their results is solid and ready.

With my contribution, I hope to create a more secure environment for company employees and a straightforward framework to connect users and security staff in a quick and resilient way.

There is only one last question.
Are humans the weakest link?
Maybe. But to quote Henry Ford, *"Don't find fault, find a remedy"*.

References

[1] D. Rock, “Managing with the brain in mind”, *strategy+business*, no. 59, 2009, Last accessed 2021/12/19. [Online]. Available: `https://www.psychologytoday.com/sites/default/files/attachments/31881/managingwbraininmind.pdf`.

[2] Proofpoint staff, “Hiding in Plain Sight - Obfuscation Techniques in Phishing Attacks”, Feb. 2016, Last accessed 2022/02/18. [Online]. Available: `https://www.proofpoint.com/sites/default/files/proofpoint-obfuscation-techniques-phishing-attacks-threat-insight-en-v1.pdf`.

[3] Experte, *Spam Checker - Email Deliverability Test*, Last accessed 2021/11/15, 2021. [Online]. Available: `https://www.experte.com/spam-checker`.

[4] MailGenius, *Free Email Tester - Email Spam Test - Email Deliverability Tester*, Last accessed 2021/11/15, 2021. [Online]. Available: `https://www.mailgenius.com/`.

[5] F. Menzaghi, “A human sensor network approach against phishing attacks”, Master graduation Thesis, Politecnico di Milano, Italy, 2020.

[6] M. Stevens, *Isolation - Mind Field (S1 E1)*, Last accessed 2021/12/07, VSauce, Jan. 2017. [Online]. Available: `https://youtu.be/iqKdEhx-dD4`.

[7] FBI Internet Crime Complaint Centre. U.S. Federal Bureau of Investigation, *Internet crime report 2020*, Last accessed 2021/09/17, 2020. [Online]. Avail-

able: `https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf`.

[8] T. Moore and R. Clayton, “An empirical analysis of the current state of phishing attack and defence”, *Workshop on the Economics of Information Security*, Jan. 2007.

[9] P. Byrd, *These are 10 of our best phishing emails. Use them.* Last accessed 2021/10/14, Jul. 2021. [Online]. Available: `https://hooksecurity.co/labs/10-best-phishing-emails`.

[10] PhishingBox, *Security awareness training & phishing simulation*, Last accessed 2021/10/14, 2021. [Online]. Available: `https://www.phishingbox.com/`.

[11] R. Publico, *How to spot a phishing website*, Last accessed 2021/06/30, GlobalSign GMO Internet, Inc., Apr. 2017. [Online]. Available: `https://www.globalsign.com/en/blog/how-to-spot-a-fake-website`.

[12] Safe Computing, University of Michigan, *Shortened url security*, Last accessed 2021/07/01, 2021. [Online]. Available: `https://safecomputing.umich.edu/be-aware/phishing-and-suspicious-email/shortened-url-security`.

[13] S. Le Page, G.-V. Jourdan, G. V. Bochmann, J. Flood, and I.-V. Onut, “Using url shorteners to compare phishing and malware attacks”, in *2018 APWG Symposium on Electronic Crime Research (eCrime)*, IEEE, 2018, pp. 1–13.

[14] D. Swinhoe, “How much does it cost to launch a cyberattack?”, *CSO Online*, May 2020, Last accessed 2021/07/01. [Online]. Available: `https://www.csoonline.com/article/3340049/how-much-does-it-cost-to-launch-a-cyberattack.html`.

[15] D. Berard, “Kaspersky lab report: The cost of a data breach continues to grow worldwide”, May 2018, Last accessed 2021/07/03. [Online]. Available: `https:`

//usa.kaspersky.com/about/press-releases/2018_kaspersky-lab-report-the-cost-of-a-data-breach-continues-to-grow-worldwide.

[16] S. Migliano, *Dark web market price index: Hacking tools*, Last accessed 2021/09/06, Top10VPN, Sep. 2021. [Online]. Available: https://www.top10vpn.com/research/dark-web-prices/hacking-tools/.

[17] Deloitte, *Deloitte puts the spotlight on the cost of cyber-crime operations in new threat study*, Last accessed 2021/09/12, Dec. 2018. [Online]. Available: https://www2.deloitte.com/us/en/pages/about-deloitte/articles/press-releases/deloitte-announces-new-cyber-threat-study-on-criminal-operational-cost.html.

[18] IBM Security, "Cost of a data breach report 2021", 2021, Last accessed 2021/07/03. [Online]. Available: https://www.ibm.com/security/data-breach.

[19] D. E. O'Leary, *What phishing e-mails reveal: An exploratory analysis of phishing attempts using text analysis*, ID 3427436. American Accounting Association, Jul. 2019. DOI: https://dx.doi.org/10.2139/ssrn.3427436.

[20] Maltego Technologies, *Maltego - Increase the speed and precision of complex investigations with Maltego!*, Last accessed 2021/11/12, 2021. [Online]. Available: https://www.maltego.com/.

[21] J. M. Hatfield, "Social engineering in cybersecurity: The evolution of a concept", *Computers & Security*, vol. 73, pp. 102–113, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.10.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404817302249.

[22] M. Huber, M. Mulazzani, E. Weippl, G. Kitzler, and S. Goluch, "Friend-in-the-middle attacks: Exploiting social networking sites for spam", *IEEE Internet Computing*, vol. 15, no. 3, pp. 28–34, 2011.

[23] H. Jhaveri, H. Jhaveri, and D. Sanghavi, "Sybil attack and its proposed solution", *International Journal of Computer Applications*, vol. 105, no. 3, 2014.

[24] M. Huber, S. Kowalski, M. Nohlberg, and S. Tjoa, "Towards automating social engineering using social networking sites", in *2009 International Conference on Computational Science and Engineering*, IEEE, vol. 3, 2009, pp. 117–124.

[25] Y. Leviathan and Y. Matias, *Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone*, Last accessed 2022/01/08, Google AI Blog, May 2018. [Online]. Available: `https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html`.

[26] C. Hadnagy, *Social engineering: the art of human hacking.* Wiley, 2011, ISBN: 9780470639535.

[27] M. Stevens, *Do You Know Yourself? - Mind Field (S1 E8)*, Last accessed 2022/01/13, VSauce, Mar. 2017. [Online]. Available: `https://youtu.be/b2ng8HuPLTk`.

[28] B. Schneier, *Essays: The Psychology of Security - Schneier on Security*, Last accessed 2020/06/18, Jan. 2008. [Online]. Available: `https://www.schneier.com/essays/archives/2008/01/the_psychology_of_se.html`.

[29] A. Tversky and D. Kahneman, "The framing of decisions and the psychology of choice", *science*, vol. 211, no. 4481, pp. 453–458, 1981.

[30] S. Grazioli, "Where did they go wrong? An analysis of the failure of knowledgeable internet consumers to detect deception over the internet", *Group Decision and Negotiation*, vol. 13, no. 2, pp. 149–172, 2004.

[31] T. Qin and J. K. Burgoon, "An investigation of heuristics of human judgment in detecting deception and potential implications in countering social engineering", in *2007 IEEE Intelligence and Security Informatics*, IEEE, 2007, pp. 152–159.

[32] K. Marett, D. P. Biros, and M. L. Knode, “Self-efficacy, training effectiveness, and deception detection: A longitudinal study of lie detection training”, in *International Conference on Intelligence and Security Informatics*, Springer, 2004, pp. 187–200.

[33] B. Schneier, *Essays: Why the Human Brain Is a Poor Judge of Risk - Schneier on Security*, Last accessed 2020/06/01, Mar. 2007. [Online]. Available: https://www.schneier.com/essays/archives/2007/03/why_the_human_brain.html.

[34] J. Haidt, “The emotional dog and its rational tail: A social intuitionist approach to moral judgment.”, *Psychological review*, vol. 108, no. 4, p. 814, 2001.

[35] H. Mercier and D. Sperber, *The enigma of reason*. Harvard University Press, 2017, ISBN: 9780674237827.

[36] M. Stevens, *The Future Of Reasoning*, Last accessed 2022/01/16, VSauce, Apr. 2021. [Online]. Available: https://youtu.be/_ArVh3Cj9rw.

[37] C. Kullenberg and D. Kasperowski, “What is citizen science? A scientometric meta-analysis”, *PloS one*, vol. 11, no. 1, 2016.

[38] SciStarter, *What is Citizen Science*, Last accessed 2022/01/07, 2022. [Online]. Available: https://scistarter.org/citizen-science.

[39] Eastern Ecological Science Center, *North American Breeding Bird Survey*, Last accessed 2022/01/07, Mar. 2018. [Online]. Available: https://www.usgs.gov/centers/eesc/science/north-american-breeding-bird-survey.

[40] A. Irwin, “No PhDs needed: how citizen science is transforming research”, *Nature*, vol. 562, no. 7728, pp. 480–482, Oct. 2018. DOI: 10.1038/d41586-018-07106-5.

[41] Australian Citizen Science Association, *10 principles of citizen science*, Last accessed 2021/06/19, 2021. [Online]. Available: `https://citizenscience.org.au/10-principles-of-citizen-science/`.

[42] Panda Cloud Antivirus, *What is Collective Intelligence?*, Last accessed 2022/02/10, 2022. [Online]. Available: `http://www.cloudantivirus.com/help/01/h_en/25.htm`.

[43] Avast Blog, *Learning Framework For Detection of Novel Malware | Avast*, Last accessed 2022/02/10, Oct. 2021. [Online]. Available: `https://securityboulevard.com/2021/10/learning-framework-for-detection-of-novel-malware-avast/`.

[44] Official Avast Support, *Managing CyberCapture in Avast Antivirus*, Last accessed 2022/02/10, 2022. [Online]. Available: `https://support.avast.com/en-ww/article/Antivirus-CyberCapture/`.

[45] A. Aleroud and L. Zhou, "Phishing environments, techniques, and countermeasures: A survey", *Computers & Security*, vol. 68, pp. 160–196, 2017, ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2017.04.006`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167404817300810`.

[46] PhishTank, *Join the fight against phishing*, Last accessed 2021/11/15, 2021. [Online]. Available: `https://www.phishtank.com`.

[47] P. Ball, *Wisdom of the crowd: The myths and realities*, Last accessed 2022/01/07, Jul. 2014. [Online]. Available: `https://www.bbc.com/future/article/20140708-when-crowd-wisdom-goes-wrong`.

[48] K. Mattingly, *The wisdom of crowds*, Last accessed 2022/01/07, TEDxBrighton, Dec. 2014. [Online]. Available: `https://youtu.be/ggUHOqq4dhw`.

[49] M. du Sautoy, *The wisdom of the crowd (with Professor Marcus du Sautoy)*, Last accessed 2022/01/07, The Royal Society, Feb. 2018. [Online]. Available: `https://youtu.be/s7tngG2kAik`.

[50] A. Lyon and E. Pacuit, "The wisdom of crowds: Methods of human judgement aggregation", in *Handbook of human computation*, Springer, 2013, pp. 599–614.

[51] Docker Inc., *Docker | Empowering App Development for Developers*, Last accessed 2022/01/09, 2022. [Online]. Available: `https://www.docker.com`.

[52] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS", in *2014 IEEE International Conference on Cloud Engineering*, IEEE, 2014, pp. 610–614. DOI: `10.1109/IC2E.2014.41`.

[53] C. Greamo and A. Ghosh, "Sandboxing and virtualization: Modern tools for combating malware", *IEEE Security & Privacy*, vol. 9, no. 2, pp. 79–82, 2011. DOI: `10.1109/MSP.2011.36`.

[54] R. Wojtczuk and R. Kashyap, "The sandbox roulette: Are you ready for the gamble", *Black Hat Europe*, vol. 414, pp. 800–125, 2013.

[55] J. Erat *et al.*, *Docker as a sandbox for untrusted code*, Last accessed 2022/01/11, Apr. 2019. [Online]. Available: `https://security.stackexchange.com/a/107853`.

[56] Docker Inc., *Docker security*, Last accessed 2022/01/11, Jan. 2022. [Online]. Available: `https://docs.docker.com/engine/security/`.

[57] D. Xanatos *et al.*, *Sandboxie-Plus | Open Source sandbox-based isolation software*, Last accessed 2022/01/09, 2022. [Online]. Available: `https://github.com/sandboxie-plus/Sandboxie`.

[58] HashiCorp, *Vagrant*, Last accessed 2022/01/11, 2022. [Online]. Available: `https://www.vagrantup.com`.

[59] Selenium, *Selenium WebDriver*, Last accessed 2022/01/13, 2022. [Online]. Available: `https://www.selenium.dev`.

[60] A. Tapper, *Playwright vs WebDriver: The Future of Browser Automation*, Last accessed 2022/01/13, Apr. 2021. [Online]. Available: `https://medium.com/slalom-build/playwright-vs-webdriver-the-future-of-browser-automation-854a7ae63218`.

[61] World Wide Web Consortium (W3C), *WebDriver BiDi - Editor's Draft*, Last accessed 2022/01/13, Dec. 2021. [Online]. Available: `https://w3c.github.io/webdriver-bidi/`.

[62] Microsoft, *Playwright Python | Fast and reliable end-to-end testing for modern web apps*, Last accessed 2022/01/14, 2022. [Online]. Available: `https://playwright.dev/python`.

[63] Zyte, *Scrapy | A Fast and Powerful Scraping and Web Crawling Framework*, Last accessed 2022/01/15, 2022. [Online]. Available: `https://scrapy.org`.

[64] PostgreSQL Global Development, *PostgreSQL*, Last accessed 2022/02/13, Feb. 2022. [Online]. Available: `https://www.postgresql.org`.

[65] M. Bayer *et al.*, *SQLAlchemy - The Database Toolkit for Python*, Last accessed 2022/02/13, Feb. 2022. [Online]. Available: `https://www.sqlalchemy.org`.