Radboud University Nijmegen

Faculty of Science

Automating Payload Delivery & Detonation Testing

Thesis MSc Computing Science

Author:
Mauk Lemmen

Student Number:
s4798937

Supervisor:
Dr.Ir. Harald Vranken

Internship supervisor:
Floris Duvekot

Second reader:
Dr Erik Poll

February 2023

Acknowledgements

I would like to express my deepest gratitude to my supervisors Floris and Harald for their feedback and guidance during this project, and to Ben and Paul for their help with technical challenges. I would also like to thank the Behaviour market group within Secura to have made my internship a very pleasant stay. Lastly but not least, my gratitude goes out to my parents for their everlasting support.

Abstract

Phishing attacks have become a significant threat to companies and organizations of all sizes, and as a result, the importance of conducting thorough phishing assessments has grown. In response to this need, we have developed a method and tool for automatically evaluating the delivery and detonation of payloads in simulated phishing campaigns. Our method employs tracking pixels to track the arrival of the payloads, with additional checks to verify edge cases. To verify payload detonation, we insert a piece of code into the payload, which proves code execution by running a command when it is opened. The output of this command, along with additional identifying information, is then communicated to a back-end server. The server processes and displays this information, providing insight into the success of the detonation attempt. This method and tool address the need for improved and automated evaluation methods in phishing campaigns. Our method provides cybersecurity companies with a valuable way to improve efficiency in their assessments.

Contents

1 Introduction

Phishing attacks are one of the most commonly used methods by cybercriminals to gain unauthorized access, spread malware, or steal information. Because of this, many cybersecurity companies offer simulated phishing attacks as a service to their clients to test their defenses.

SpecterOps, one of these companies experienced in performing phishing attacks, has written a blog post[1] outlining where the current industry standard for phishing campaigns often falls short. For example, they note that the outcome of a phishing attempt is almost always already known: the client can be phished. Even a 1% click rate of malicious content inside a phishing email is enough. Therefore, it is of little value to show the client that they can be phished, and it would be more beneficial to focus on what would happen when they get phished.

SpecterOps proposes a new approach to evaluating a client's susceptibility to phishing that relies on close collaboration between the security analysts executing the phishing and the client's internal security team. This approach enables a more direct understanding of what the client wants out of a phishing assessment, e.g. mainly focusing on detecting phishing attacks, or dialing in on risk mitigation after having been phished, etc., which results in better recommendations and outcomes with meaningful metrics. SpecterOps identifies four main topics, each with its own metrics and methods of assessment, that are key to understanding the client's risk level when evaluating the client's susceptibility to phishing with this new approach.

1. **Social Engineering:** What percentage of my user base is susceptible to phishing and will engage the phisher or click on the 'evil' parts of a phish?

2. **Payload Delivery:** What types of messages and attachments will successfully land in my users' inboxes?

3. **Payload Detonation:** What payloads will successfully detonate on my users' systems?

4. **Response Process:** Is my team able to detect and respond to a successful phish?

Questions one and four have straightforward approaches. With regard to the first question, a multitude of products exist to conduct phishing campaigns and measure statistics about how many times the phishing email was opened, how many times credentials were entered, and more similar metrics. These tools are currently part of any phishing simulation by default. Answering the fourth question is part of standard operating procedures as well: did the security team notice and respond to the simulated phishing attack? However, questions two and three are more difficult to assess. SpecterOps notes the lack of existing public tools to help answer these questions. Cybersecurity company Secura, with whom this thesis is in collaboration, is interested in the creation of such tools and wishes to explore SpecterOps's approach further.

When trying to answer these questions for a client, a cybersecurity company may encounter several challenges, one of which is the need to establish a consistent method for testing the security aspects in question. To go from the questions of "What types of attachments will land in the clients inboxes" and "What payloads will successfully detonate on my users' systems" to the actual answers, some method is needed that describes how one can test for and obtain that information. This methodology should not only be reliable but also efficient, ideally incorporating automation for practicality within a corporate setting. No such methods currently exist to the satisfaction of Secura, either in an academic sense or in a practical sense.

Therefore, the main questions of this research are:

- How can the delivery of payloads be tested (automatically)?
- How can the detonation of payloads be tested (automatically)?

To answer these questions, we design and develop a testing environment that is able to automatically and remotely assess whether a series of potentially harmful payloads manages to get through the email firewall of the client. Additionally, the environment is able to remotely assess whether a harmful payload is able to execute its malicious code or is stopped by an antivirus system when opened on the client's system.

The challenges of answering each of the research questions are explained in more detail in their separate sections below. Additionally, for the remainder of this thesis we use two distinct roles in order to clearly communicate the methods and environment being developed. The first role is that of the security analyst, who is a member of a cybersecurity company and is responsible for using the method to test the security of the target. The second role is that of the client (or target), which refers to the company, or individual that the security analyst is performing the test on. These roles will be used throughout the remainder of the paper in order to clearly describe the method and the results of the study.

1.1 Payload Delivery

The goal of this part of the research question is to be able to automatically and remotely test confirmation of delivery for multiple payloads during a phishing campaign. Firewalls may block out certain attachment types or detect malicious code in a file. A client needs to be able to tell which types of attachments were successfully blocked, and which managed to bypass their detection systems.

The challenge of developing an environment for payload delivery testing can be broken down into three parts. Firstly, the security analyst needs to be able to send a collection of payloads to the client in a way that simulates an email by an attacker. The security analyst cannot just include all attachments to one email, write a random email body and send it, as there are many factors to take into account that determine if an email is allowed through a firewall. Solving this problem requires the insight into how email works, how payloads could be transported via email, and an engineering solution to facilitate the desired functionality. Secondly, a technique is required to confirm that the email with attachment actually arrived in the inbox. Manually checking each email at the target's location is trivial, but doing it automatically, remotely, and on a large scale requires a more thought out method. Solving this issue requires an understanding of how emails can be tracked, how this can be applied to our situation, and the development or use of a tool to implement the solution. Lastly, the confirmation of arrival for each email that has been sent out needs to reach the security analyst and be presented in a readable manner. Solving this challenge requires an engineering solution to use the result from the email tracking and transforming that into a clear overview for the security analyst.

To summarize, for the payload delivery section of the research question, we research the following three sub-questions and develop an environment with the resulting answers:

1. How should the required payloads be sent to the target?
2. How should the delivery of the payloads be confirmed?
3. How should confirmation of payload delivery be communicated to the security analyst?

1.2 Payload Detonation

The goal of this part of the research is to be able to automatically and remotely measure and confirm payload detonation for multiple payloads during a phishing campaign. It is very possible that an email with a malicious attachments slips through the email firewall and is opened by the receiving user. In that case, it is important for the client to know if the defensive measures that are in place on their users' workstations are effective at stopping the malware from executing or not.

Similarly to payload delivery, the challenge of creating a method to test payload detonation can be divided into three parts. Firstly, a number of payloads need to reach the target. Since the premise of this part of the research is that a malicious payload has slipped by the email firewall and is opened by an unsuspecting employee, simulating an attacker in the sending of the email is less of an issue, as it is irrelevant in such a case. Nonetheless, the payloads need to arrive at the target without being blocked by the email firewall, so it is a necessary question to answer. Secondly, a (preferably automated) mechanism is required to measure if a payload manages to execute its code on the target system. Solving this problem requires a method to determine whether code was executed, as well as a technical implementation to achieve this. Lastly, the detonation confirmations need to be communicated back and displayed to the security analyst. This means that a form of communication needs to be established with the remote security analyst by the executed payload. To achieve this, a literary review is required of techniques used to exfiltrate information and methods that are commonly used to detect and prevent unauthorized remote communication. Subsequently, an exfiltration technique chosen from the review is used for the technical implementation of this functionality. To summarize, for the payload detonation section of the research question, we research the following three sub-questions and develop an environment with the resulting answers:

1. How should the required payloads be sent to the target?
2. How should the detonation of the payloads be measured?
3. How should the detonation of the payloads be communicated the security analyst?

1.3 Reading Guide

In the remainder of the thesis, we first dive into background information that is necessary to understand the concepts used in our research. Secondly, we discuss the research design, which includes the general approach that was taken to answer the research questions, as well as the requirements for the approach and its implementation. Following that, the theoretical design of the payload delivery and detonation environment is explained by answering each of the research questions, ending with the technical implementation of the environment and conclusion of the entire research.

2 Background

This section aims to provide background information that is necessary for understanding the terms and concepts used throughout the thesis. Firstly, email and its components are explained, as well as the protocols that are used to authenticate emails and protect them against spoofing. These protocols are relevant for the email server that is needed for the environment that is developed in this thesis, which can be found described in more detail in Chapter 3. Following the information about email, an explanation of phishing and the phishing process is given, as well as an explanation of maldocs that may be used as payload in a phishing attack.

2.1 Email

Email is a widely-used method of electronic communication in which messages, typically containing text, files, or images, are sent and received over a network. The process of sending and receiving email involves several key components and protocols, including the Mail User Agent (MUA), the Mail Transfer Agent (MTA), and the Simple Mail Transfer Protocol (SMTP)[2].

The MUA, also known as the email client, is the software that users interact with to compose, send, receive, and manage their email messages. Examples of MUAs include Microsoft Outlook, Apple Mail, and Gmail.

The MTA is a server-side program that is responsible for transferring email messages from the MUA to the recipient's MUA. MTAs communicate with each other using the SMTP protocol, which is a set of rules for sending and receiving email messages over a network.

Once an email message has been delivered to the recipient's MTA, it is typically stored on the recipient's email server until it is retrieved by the recipient's MUA. There are two main protocols for retrieving email messages from a server: the Internet Message Access Protocol (IMAP)[3] and the Post Office Protocol (POP3)[4]. IMAP allows users to access and manage their email messages on the server, while POP3 downloads the messages to the user's computer and removes them from the server.

2.1.1 Email Format

An email [5] message consists of two parts: the header and the body.
The header of an email encompasses a number of header fields, each of which has a name and a value separated by a colon symbol. The message header must include at least the "from", "to" and "date" fields, and may include many other fields registered at the Internet Assigned Numbers Authority (IANA). The "from" field commonly contains the email address of the sender, but it should be noted that any address can be filled in there, thus it is possible to spoof an

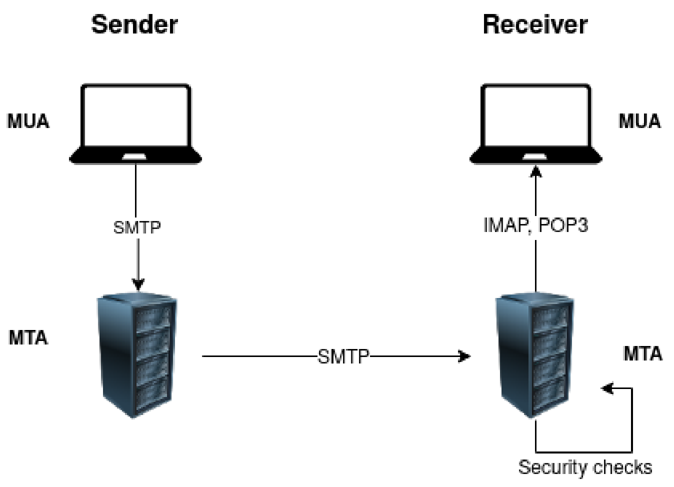


Figure 1: Schematic of the architecture of sending an email

email as if it has been sent by another person. Contrarily, the “to” field must contain the correct email address of the recipient, otherwise it will not arrive. However, it is possible to include a name in this field as well, which does not have to be correct. Other common headers include:

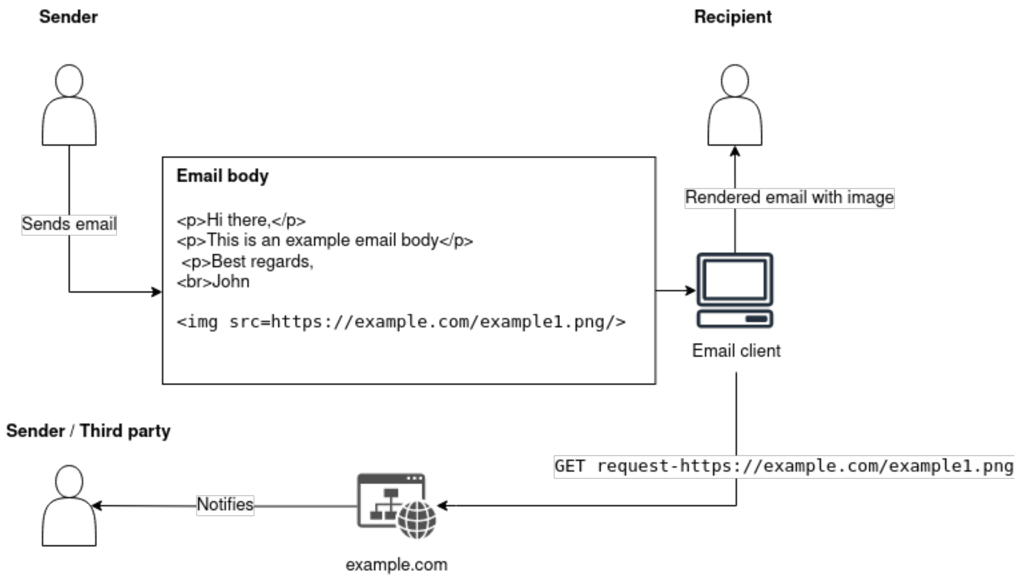
- Subject: topic of the message
- Cc: carbon copy, additional recipients
- Bcc: blind carbon copy, only included in the SMTP delivery, not in the message itself.
- List-Subscribe: contains the URL to get a subscription to the mailing list from which this message was relayed.
- ...

One specific header, relevant to this research, is the Return-Path. When the email leaves the SMTP environment at delivery to the final MTA, this header is included, containing the email address with the domain that the email was sent from. This domain will be used by the recipient’s MTA to perform various security checks on related to SPF, DKIM, and DMARC and spam detection. It is noteworthy that this domain does not have to match the “from” header field. Whilst the “from” field can be spoofed with any email address, the return path will need to match the domain specified in the SPF, DKIM and DMARC records to pass these security checks.

The body of an email is the main content area of an email message where the sender can include text, images, and other multimedia elements. The body of an email is typically displayed below the subject line and sender information, and above any attachments or signature that may be included. Formatting options, such as font size, color, and style, can be used to make the text more visually appealing and easier to read. Hyperlinks to other web pages or documents can also be included to provide additional information to the recipient. In addition to plain text, the body of an email can also include HTML or other markup languages, which allow for more advanced formatting and layout options.

2.1.2 Email tracking

HTML and remotely hosted content in emails can be used to track emails through HTTP requests.[6] This is done through the use of personalized URLs. For example, if an email contains an image hosted on a server, and that email is opened, an HTTP request is made to the server to retrieve the image. If the image is a commonly requested image and not specifically tied to a single email, the server cannot distinguish which request belongs to which email address. However, if the image has a unique URL that was created specifically for that email, the server can log the request and determine that the email was opened. An example of how image tracking can be performed can be seen in Figure 2.



```
<p>Hi there,</p>
<p>This is an example email body</p>
 <p>Best regards,
<br>John

<img src=https://example.com/example1.png/>
```

Figure 2: Schematic of how a tracking image works

2.1.3 DKIM, SPF and DMARC

DKIM (DomainKeys Identified Mail)[7] is a method of authenticating email messages using a digital signature. This signature is added to the email headers and can be verified by the recipient's email server. The purpose of DKIM is to prevent email spoofing, where an attacker sends an email that appears to come from a legitimate sender, but is actually from a different source. The DKIM signature is generated using a private key that is held by the sender's email server. The corresponding public key is published in the sender's DNS records, allowing the recipient's server to verify the signature. The signature includes information about the sender's domain, as well as the email message itself. If the signature is verified, it means that the email was sent by an authorized server for the sender's domain, and that the message has not been altered in transit.

SPF (Sender Policy Framework)[8] is another method of preventing email spoofing. With SPF, a domain owner can specify which IP addresses are allowed to send email on behalf of their domain. This is done by adding a special record to the domain's DNS records, which lists the authorized IP addresses. When an email is received, the recipient's server can check the SPF record to see if the sending IP address is authorized. If the IP address is not authorized, the email may be flagged as suspicious.

DMARC (Domain-based Message Authentication, Reporting and Conformance)[9] is a system that builds on the concepts of DKIM and SPF. It allows a domain owner to publish a policy in their DNS records, specifying which mechanisms (such as DKIM and SPF) are used to authenticate their emails, and what to do if an email fails authentication. This policy can include instructions for the recipient's server to reject or quarantine the email if it fails authentication. Additionally, DMARC provides a mechanism for reporting on the effectiveness of the domain's authentication policies. The domain owner can specify an email address to receive reports on the number of emails that pass or fail authentication, as well as any actions taken by the recipient's server based on the domain's policy. This can help the domain owner identify any potential issues with their authentication settings and make adjustments as needed.

Altogether, these three tools provide a system for verifying the identity of email senders and protecting against email spoofing and phishing attacks. By implementing DKIM, SPF and DMARC, a domain owner can help ensure that their emails are not easily spoofed, and that recipients can trust the authenticity of the messages they receive.

2.2 Phishing

2.2.1 Definition

The study by Alkhalil et al., 2021[10] defines phishing as *a socio-technical attack*, in which the attacker targets specific valuables by exploiting an existing vulnerability to pass a specific threat via a selected medium into the victim's system, utilizing social engineering tricks or some other techniques to convince the victim into taking a specific action that causes various types of damages."

2.2.2 Phishing process

Alkhalil et al. note the existence of several studies (Rouse, 2013)[11], (Jakobsson and Myers, 2006)[12], and (Abad, 2005)[13] with a different division of phases to describe the phishing process. However, they propose a new anatomy with a more detailed look at phishing attacks, and that is the model we reference to as well. Four phases are referenced: the planning phase, attack preparation phase, attack conducting phase, and valuables acquisition phase. We briefly describe each below.

The planning stage is the first phase of a phishing attack, during which the attacker gathers information that can be used to exploit psychological vulnerabilities, which refers to ways in which individuals can be manipulated or deceived by psychological means. This information includes data like names, email addresses, social media information. Additionally, this stage includes the building of fake websites, creation of malware and design of phishing emails.

In the attack preparation phase, the attackers look for exploitable vulnerabilities. These vulnerabilities can range from exploits in applications that the target uses, to unsafe browsers, to zero-day vulnerabilities that are still publicly unknown. During this stage, attackers also determine the medium over which the attack will be carried out.

During the attack conducting phase, the phishers carry out the prepared attacks from the previous phase. Based on the victim's interaction with the attack, followup attacks may be conducted to further compromise the target's systems.

In the valuables acquisition phase, the last phase of the proposed phishing life cycle,the attacker collects valuable information from the target and uses it for their sinister purposes, such as the theft of money or the sale of credentials on the black market.

2.3 Maldoc

A maldoc (Malicious Document) is a type of malware. It consists of a document, such as a PDF or Microsoft Word file, and a piece of embedded malicious code. Depending on the type of maldoc, the malicious code executes either when the document is opened, or when the victim grants certain permissions to the document. Typically, a maldoc arrives on a victim's computer through a phishing email, but there are many other types of delivery methods that can be used. In the following section, we explain the contents of maldocs, and the considerations that go into their creation, as defined by security experts within Secura.

2.3.1 Maldoc Payload

File Type
There are two categories of payload files. Executable files, which directly execute the payload, or container files, which contain the executable file. Such a container file may be used to bypass defense methods.
For a Windows target, executable file types can come in the form of:

- application files (.EXE, .MSI, ...)
- scripts (.BAT, .CMD, .JS, ...)
- shortcuts (.LINK, .URL, ...)
- code execution supporting documents (.CHM)

When targeting Microsoft Office, executable file types come in the form of:

- Macro-enabled documents (.DOC, .DOCM, PPT, PPTM, XLS...)
- Add-ins (.XLL)

Container file types for a Windows target are:

- Archive files (.ZIP, .CAB)
- Disk images (.ISO, .VHD, .IMG, ...)
- Documents (.RTF, .HTML)

For Microsoft Office, a container file type exists in the form of OLE objects. This object allows object linking and embedding to documents and other objects.

Execution method
The consideration for execution is between a staged and stageless payload. If the payload is stageless, it will contain all of the necessary code to fully execute its goal. The benefit of this approach is that no extra downloads are needed after the original delivery. This means that the payload has full control over all of the network traffic that it generates. If the payload has the capability built in, it could obfuscate its presence by emulating benign programs, as well as further hide suspicious network activity by encrypting its communications. The downside to this approach is that a larger-sized payload is required, which

consequently needs to be able to evade detection.

With a staged payload, an initial smaller file or command is used to download the code for the next stage in the attack. The benefit of this method is a smaller payload size, often allowing a one-liner command as the original payload. This comes with the added convenience that later stages need not necessarily be written to disk, thus avoiding antivirus disk scans. The downside of this approach is more uncontrolled network traffic, as well as more possibilities for information to leak during the downloading and execution of further stages.

2.3.2 Maldoc countermeasures

Defensive measures exist both on host level and network level. Both of these can hinder the payload in executing its intended task. Below, several hurdles are listed that need consideration when a maldoc is constructed.

Host-based defenses
Host-based defenses are security measures present on the target host device. We note three categories of ways these could hinder the payload.

- **Detection at rest**: The system may detect the payload file as malicious. This comes in the form of antivirus scans and the presence of a suspicious file type.
- **Execution Restrictions**: The system may prevent untrusted files from executing. For example, files downloaded from the internet may have a *mark of the web*, marking it as potentially unsafe.
- **Detection during execution**: The system may detect suspicious behaviour when the payload executes. Antivirus software may match the execution of malicious code with malware signatures in its database.

Network-based defenses
Network-based defenses are security measures present on the network that the target is connected to.

- **Detection at payload delivery**: The delivery attempt of the initial payload, or the payload of later stages may get detected.
- **Detection of command & control traffic**: Traffic pertaining to communication between the malware and attacker may get detected.

3 Research design

3.1 Approach

With this research, the goal was to both research and design a theoretical environment that encompassed the answers to the research questions, as well as a practical implementation of this environment that could be used by Secura. In order to develop the design for such an environment, we transformed each of the research questions into sub-questions and requirements: the sending of the payloads, confirmation of delivery or detonation of the payloads, and the communication of the results to the security analyst. Additionally, we identified the components and assumptions that would be necessary for the environment to work, which we have described in more detail later in this Chapter.

For payload delivery, we first evaluated the ways in which form payloads could be sent to the target and then determined which should be included in a payload delivery testing environment. Secondly, we looked at how the delivery of payloads could be confirmed to have arrived at the target. The method that we settled on was the use of a tracking pixel inside of the emails, which is a type of tracking is typically used in advertisement campaigns and similar type of operations. In addition, a manual backup method was established in case the pixel-tracking method would be hindered. Thirdly, to get an accurate result of delivery confirmation back to the security analyst, we analysed cases where the pixel tracking method could fail, and devised a list of file extensions that need an extra confirmation to verify their arrival after the tracking pixel has triggered.

For payload detonation, we also first established the way in which the payload should be delivered to the target, but as the focus of payload detonation lies on the measuring of the detonation after the payload has already arrived at the target, the method of delivery was trivial: via email or via an online download link sufficed. To measure the detonation of a payload, we first established that the goal of payload detonation testing is to determine if any code execution took place. With that principle in mind, we devised the idea that a payload should execute an arbitrary code command, which when being able to execute, would be proof of code execution. Lastly, we looked at an automated way to get the result of code execution on a target system back to the remote security analyst. For this, we researched data exfiltration techniques and their countermeasures in literature, and concluded two viable options for the extraction of the payload detonation execution result: via DNS or via HTTP(s).

The technical components and explanation of the implementations of both the payload delivery and detonation environment can be found in Chapter 5.

3.2 Setup

As the research setup, we assumed access to an email server successfully configured with DMARC, DKIM and SPF. As explained in earlier chapters, these security measures have great relevance in the decision of emails being detected as spam or not. Any sophisticated phisher will have access to such an email server, so this is the degree of attack sophistication we expect a client to deal with as well. This includes a configured DNS server with DMARC, DKIM and SPF records. Without these enabled, the validity of the accuracy of the payload delivery test results cannot be guaranteed, as all emails may get blocked by default because of insufficient security protocols enabled.

For both the testing of payload delivery as well as payload detonation, we assumed collaboration between the security analysts performing the phishing and the internal security team of the client, as described by Specterops. For the payload delivery aspect, this means that a test email account has been provided by the client. All emails with payload are sent to this account, and are to be opened on arrival. If a content-block warning is given about the body of the email, it should be disabled and all content should be allowed, as otherwise the tracking of the emails may be hindered. Such a content-block warning has no relevance to the arrival of the attachments themselves. Any email body used is assumed to have been tested without payload, to ensure any blocking will be because of the payload itself.

For payload detonation, we assumed that a test email account as well as a workstation is provided by the client, which should be configured just how a regular employee's would be. Security controls should be configured such that all inbound emails from the security analyst are let through, as only payload detonation is relevant here and payload delivery is separately tested. On email arrival, the person operating the workstation must download the attached payload to their system and open or execute it, depending on the type of payload. If the payload in question is an executable file type, it should be executed and any following warnings discarded and disregarded. In the case of a Microsoft Office file, when prompted, macro content should be enabled. If the payload is a container file type, it should, if relevant, be extracted and the container contents should be executed or opened similarly to previously described executable file types.

4 Design

4.1 Payload Delivery

For this part of the research question, we assume the target has supplied a test email account to send emails to, as well as either a contact at the target who is available to open incoming emails, or direct access to the email account by the security analyst themselves.

4.1.1 Sending the payloads to the target

In order to test whether a collection of payloads arrives at the target, the first step is to be able to create and send emails containing them. Firstly, a sending address needs to be chosen. This address will be what the recipient of the emails sees as the source address, which should match the domain that is owned by the security analyst, otherwise it may fail a DKIM verification check. Naturally, should the security analyst want to verify whether the email server has implemented these security checks, a sending address from a different or non-existing domain can be chosen. Next, it is imperative that the body of the email does not get detected as spam by the recipient's MTA and therefore blocked or quarantined, because this would make it impossible to determine if the email was blocked because of the attachment or the body. Additionally, the email would preferably not land into the spam inbox of the MUA either, but this problem can be remedied by adding the source email as a trusted sender. Finally, for the attachments themselves, there are three options depending on what the security analyst wishes to test:

- Directly attach the payload as email attachment. If the email with this attachment would get blocked, that would indicate that either the payload was detected as malicious, or that the attachment type is blocked as a firewall rule.
- Encapsulate the payload in some sort of container file, such as a *zip* file or *iso* file, and attach that container as email attachment. This will allow for the determination of whether the defensive systems scan the contents of the container and block any malicious items in it.
- Don't attach the payload as an attachment, but upload it to a hosting service and include a download link to the file in the email body. This option does not test the email defense systems related to the payloads, but it does determine whether these type of links are allowed and let through in the email body. While this is a useful and interesting test, it is out of scope for the goals of this research, as we aim to develop a method to test the delivery of the payloads themselves through the email firewall, not to detect which file hosting services are blocked.

We see both the first and second option as requirements that should be implemented in the delivery testing environment. This will allow for both an assessment of the scanning mechanisms with direct payloads as attachments, as well as reveal whether the target's defense system does a deeper scan on container files. The actual technical implementation of how the emails themselves are transmitted to the target is discussed in Chapter 5.

4.1.2 Confirming delivery of the payloads

To confirm the arrival of payloads at the recipient, we propose the use of a tracking pixel.[6] A tracking pixel is a link to an image, usually 1-by-1 pixel large that is inserted into an email. When the email is opened, the browser will process that link and make a web request to retrieve the image content, at which point the website hosting the image can log the incoming request, thus knowing if the email was opened or not. In the case of this research, a separate tracking pixel can be used for each email containing a different attachment. This method will allow for confirmation of which email has been received and opened. The HTML code for a tracking pixel could look like:

```
<img src="https://some-website.com" width="1" height="1" border="0" />
```

It is possible that general anti-tracking countermeasures are active on the MUA and that, therefore, HTML content is blocked by default. In those cases, if the option is present, email-content should be set to "allowed" for that specific email or the user should be added to trusted contacts, such that HTML content is no longer blocked, and the tracking pixel works.

One possible downside to this method is that antivirus checks by the recipient's MTA may activate the tracking pixel while not actually delivering the email. In some instances, such as with Microsoft Outlook, emails may be passed on to the recipient while only blocking the attachment. This can make tracking the delivery of the attachment less effective because the pixel will still trigger even if the attachment was blocked but the email itself was not.

As a backup method, the name of the attachment can be included in the body of the email. This way, when all tracking methods fail, the names of payloads that managed to get through can be manually confirmed by either the security analyst with access to the test account, or by the client themselves. Alternatively, the logging files of the email server could be checked to see which emails have been delivered or not. However, this would require access to the client's email server, which would likely need to be accessed by a technical person with clearance at the target company. This is a hindrance that can be avoided with the first method of going through the arrived emails, which is something that can be performed by a person with no technical skills or clearance.

4.1.3 Communicating delivery confirmation to the security analyst

Given the assurance that each email on the test account has been opened, either by the security analyst or by the client, the delivery confirmation can be automatically communicated through the tracking pixel. The HTTP request made by the tracking pixel is logged by a listening server, and a script can check these logs and add the information to the list of delivered attachments.

However, because of the issues mentioned regarding the tracking pixel, additional steps need to be taken to ensure a correct representation of the situation to the security analyst. We propose the use of a list of file extensions typically stripped by Microsoft Outlook[14] and Google Mail[15] to track which payloads need extra confirmation. When the server is notified that an email has been opened, it can then first check if the file extension of the email's attachment is in the list of blocked extensions, and if it is not, then the payload can be marked as *delivered*. However, if it does match with an extension from the list, it can instead be marked as *maybe_delivered*.

After all emails have been opened, the security analyst can go through all payloads marked as *maybe_delivered* and compare them with the received emails to confirm if those attachments were actually received or not and manually mark them as *delivered*.

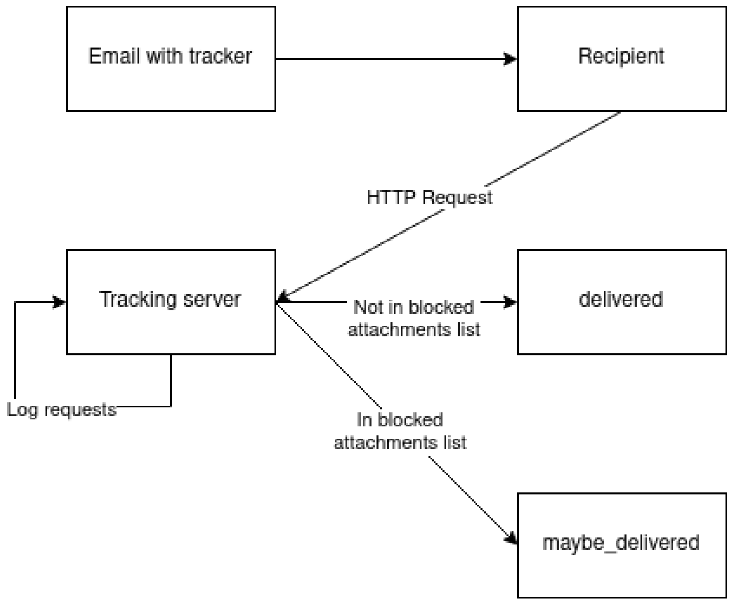


Figure 3: Schematic of the tracking process

4.2 Payload Detonation

As mentioned in Chapter 3, we assume a collaboration between the security analysts (attackers) and internal security team where a test email account as well as a workstation has been provided by the client, which should be configured just how a regular employee's would be. Security controls should be configured to allow inbound emails from the security analysts, and to accept any prompts to enable-web-content for received emails.

4.2.1 Sending the payloads to the target

Security controls have been set to allow inbound emails from the security analyst, so it is not necessary to craft a specific "inconspicuous" email. Instead, the payloads can be sent one by one with a nondescript email body, or they can be sent using an online file hosting platform or any other web-based method. It is important to note that operating systems may apply a "mark-of-the-web" to files downloaded from the internet, which can cause antivirus software to display specific behavior. This behavior also applies to attachments downloaded from emails, so it so it should be made sure to be taken into account in the payload detonation testing environment.

4.2.2 Measuring detonation of the payloads

To be able to test a variety of payloads, we require each payload to execute the same code. The payload opens a shell and executes a simple *whoami* command, which displays the current user active user. This information is then saved to a local file together with the type of file that the payload was in, ready for communication back to the security analyst's server. By having executed this code and receiving the resulting information, it has been established that it was possible to execute code on the target system, thereby implying that an attacker has gained at least user level access to the device.

It may also be the case that a security analyst wants to test if specific code manages to execute, or gets blocked by an antivirus. In those cases, that specific code simply needs to be executed before the *whoami* command. Any additional information may be appended to the file that gets communicated back to the server. Different types of payloads may have unique methods of getting code execution, but the manner of how that is achieved is irrelevant, as long as the *whoami* command is executed and the payload's filetype, name, and *whoami* command result are communicated to the server. We propose the following format to contain all necessary information:

```
filetype: [type], filename: [name],
machine: [machine], additional: [info]
```

4.2.3 Communicating detonation to the security analyst

The proof of payload detonation needs to be communicated to the security analyst, which means that it must either be transmitted over the internet to the analyst, or shown locally to the person executing the payloads. Ideally, the payload would be transmitted over the internet, but this may not be possible depending on the firewalls in place. We do not consider advanced data exfiltration within scope of this research, merely as a method to help automate confirmation of detonation. Defensive measures constantly evolve, so any specific exfiltration technique might have a short shelf-life. Nevertheless, it is preferable that some form of communication between the target computer and a server managed by the security analyst is achieved. In order to find a good method for the extraction of the information, we have gathered a list of relevant defensive countermeasures that need to be taken into account. In the paper by (Ullah et al.)[16] a wide range of data exfiltration attack vectors and defensive countermeasures against these attack vectors is reviewed and summarized. According to the paper, payload detonation testing is a type of data exfiltration vector that falls under the *Spyware and Malware* category, for which a specific set of countermeasures is relevant. The authors distinguish between preventive and detective countermeasures, which focus on proactively resisting and reactively detecting data exfiltration attacks, respectively. However, most of the preventive countermeasures discussed in the paper are not applicable in this case because our proposed payloads do not access pre-existing sensitive files on the target system. In Table 1, we have extracted all relevant countermeasures that need to be considered when deciding on an exfiltration method.

Paper	**Contribution**
Al-Bataineh & White[17]	Leverages encryption as an opportunity instead of a challenge for detection of data exfiltration.
Peneti et al.[18]	Attaches a time stamp with each document and later monitors the time stamp of each outgoing document. If time stamp of outgoing document is old, so it is allowed to leave else transmission is blocked.
Koch & Rodosek[19]	User features are extracted from network traffic and compared with already created user's profiles. Any user whose profile does not match with already existing profiles is considered an attacker.
Berlin et al.[20]	Detect malicious behaviour based on analysis of windows logs.
Rajamenakshi & Padmavathi[21]	Framework for gathering network and host data and analysing it for detecting data exfiltration.

Table 1: Table of defensive measures

Al-Bataineh & White: Identifies and uses the following anomalies to detect malware traffic:

- Repeatability of issuing HTTP GET and POST requests
- Type-mismatch of content between the declared type in HTTP header and the actual content
- Encryption and compression of POST request content
- Embedding of encrypted commands in the body of GET requests and POST responses

Possible solutions:

- Avoid repeating HTTP requests.
- Correctly declare the type in the HTTP header.
- Avoid encrypting the POST request content.
- Avoid the use of HTTP(S) to send the data.

Peneti et al.: The solution presented in this paper uses timestamps and confidentiality scores to determine whether a document should be allowed or blocked from transmission. If the document has a recent timestamp, its confidentiality score will be calculated. If the score is above the specified threshold, the document may be blocked from transmission.

Possible solutions:

- Avoid confidentiality detection by limiting confidential information as much as possible.
- Don't use files, but rather direct transmit information.

Koch & Rodosek: The described system creates user profiles based on network traffic in an encrypted environment and compares them to existing user profiles. A "Command Evaluation" module is used to detect malicious behavior by identifying attacker commands in malicious sessions.

Possible solutions:

- Use commonly used commands to communicate the data, while avoiding commands that might identify attackers.
- Avoid the use of tunneled protocols that may be under scrutiny by detection programs.

Berlin et al.: This paper presents a supervised machine learning method trained on Windows logs to detect known malicious behavior. However, the approach may not be effective in detecting newer, unknown malware.

Possible solutions:

- Use a method with atypical behavior for malware.
- Use a method that is not considered to be malicious behaviour.

Rajamenakshi & Padmavathi: The proposed system uses an exfiltration detection engine that compares behavior obtained during the training phase with behavior during the detection phase. Anomalies in internet traffic is compared with deviations in behavior for other categories such as CPU usage classify anomalies as possible infiltration of exfiltration.

Possible solutions:

- Try to use commands that are typically used on a system to avoid anomaly detection.

Chosen solution

Based on the prevalent defensive measures shown in the papers, we have derived two methods of communication for the payload detonation confirmation. The first method is the use of existing software to make a HTTP or HTTPS POST request. Despite the frequent targeting of these protocols by the anti-exfiltration measures, they can still be a useful choice. Firstly, the use of HTTP(S) is so commonplace on target devices, that the outright blocking of them is rare. Secondly, the actions required to communicate the detonation confirmation are not malicious or of association with typical malware. Therefore, it is relatively easy to avoid the detection of software looking out for such malicious activity. Lastly, by making use of tools already present on the target system, commonly referred to as "living of the land" attacks, it is possible to blend in even further, as these tools tend to be more trusted than freshly downloaded software or code. One example of such a tool is *curl*, which is a command line tool used to transfer data, by default part of current Windows systems. *Curl* can make both HTTP and HTTPs requests, including the sending of files using POST requests.

An example macro for Microsoft Word which performs the desired task:

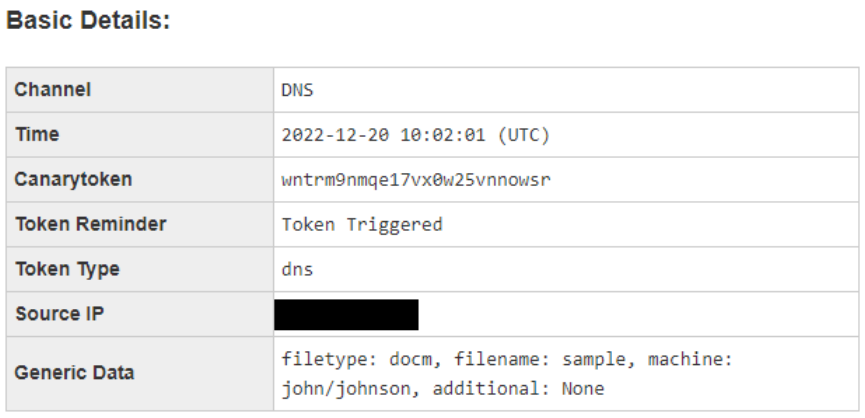
```
Sub AutoOpen()
MyMacro
End Sub
Sub Document_Open()
MyMacro
End Sub
Sub MyMacro()
Dim Str As String
Str = "cmd.exe /c echo filetype: docm, filename: example,
    machine: > %tmp%/info.txt &&
    whoami >> %tmp%/info.txt &&
    curl.exe -d ""@%tmp%\info.txt"" server-address"
CreateObject("Wscript.Shell").Run Str, 0, True
End Sub
```

Figure 4: VBA script that executes the *whoami* command and sends the result together with additional identifying information to a server

The second method is the use of DNS to exfiltrate the information. Due to the important role of DNS in corporate environments, DNS traffic is often let through firewalls. An attacker can make use of DNS to exfiltrate data by trying to resolve a subdomain of a self-hosted DNS server on the target system. However, instead of trying to resolve a legitimate subdomain, the attacker can insert a piece of data in place of the subdomain such that the DNS server receives a request to resolve that piece of data instead. Because the DNS server is hosted by the attacker, the request can be logged, and the data saved. If the attacker wants to exfiltrate more data than fits in a hostname (255 bytes)[22], the data has to be split up and sent in consecutive DNS requests. One example of a software that can accomplish this is *Canarytokens*[23]. The paper (Gionathan Armando Reale, Benjamin Zinc Loft, 2019)[24]) about Canarytokens mentions the limitation that certain document readers or scanners may trigger the token (prematurely) or not trigger it at all. However, because our proposed method sends along the result of the *whoami* command as well as some additional system information, any DNS resolve requests to the server without the additional data can be ignored.
The following is an example of a windows command that transmits some encoded data to the security analysts DNS server:

```
nslookup [encoded-data].[token-code].analyst-server.com
```

In the case of a Canarytokens server, when a DNS token triggers with the payload detonation data attached, the server will notify the user through a web-hook or email with information about the trigger.



Basic Details:

Channel	DNS
Time	2022-12-20 10:02:01 (UTC)
Canarytoken	wntrm9nmqe17vx0w25vnnowsr
Token Reminder	Token Triggered
Token Type	dns
Source IP	[redacted]
Generic Data	filetype: docm, filename: sample, machine: john/johnson, additional: None

Figure 5: Result of a Canarytoken triggering with additional data

5 Implementation

In this section, we will delve into the implementation of the method discussed in the previous section.

5.1 Payload Delivery

The process of testing payload delivery involves two types of components: active components with user interface, such as the Python program and Gophish, and passive components with background functions, such as a DNS server, email server, and internet traffic relay. The active components were the focus of the implementation and will be discussed in more detail, while the passive components are relatively standard and will not be explained in depth.

5.1.1 Active components

Gophish[25] was chosen as the program to facilitate the sending of emails as well as the tracking of which emails were opened. Gophish is an open source framework used for simulating phishing campaigns. It works through a system of email templates, describing the body of the email and any attachments, which are then used in campaigns that target groups of users. Additionally, these campaigns can be configured to support the use of fake pages, tracking images and links to further simulate a real phishing attack. Metrics such as the amount of times a link has been clicked, or how many people entered data on the fake phishing page can be tracked from Gophish's dashboard. For our purpose, due to the way Gophish requires campaigns to be created, it would be infeasible to use for the goal of sending multiple different attachments to the same testing email address. Therefore, we have written a python program which communicates with Gophish through its JSON API.[26] This python program can fully manage the payload delivery testing process, including the creation, sending and tracking of emails to the target.

Firstly, we created the possibility to generate templates for Gophish, which each consist of one attachment, the HTML source page for the email, the subject of the email, and the email address from which the email should be sent. We added 35 sample attachments chosen from the list of maldoc file types as described in Chapter 2.3.1 and grouped them based the type of attachment, such as executables, Microsoft Office macro files, container files and more. During the generation of the templates, either all or one of the groups of attachments can be chosen to create templates with, depending on the specific needs of the test that is to be conducted. Additionally, custom folders of attachments can be created if that is required.

Secondly, there is the option to create and launch campaigns using previously created templates. When this option is selected, the target recipient for the emails can be set, as well as the launching date, and the delay between emails.

Once these are set, the campaign is launched and can be visually tracked through the Gophish user interface or by keeping track of the results showing in a _results.csv file that holds the current project's information and the delivery status of each attachment. Examples of this file can be found in Appendix A.1

Thirdly, the security analyst can choose the option to check for finished campaigns. It does this by assessing in which campaigns Gophish has noted the email being opened because the tracking pixel was triggered. Once it finds such an email, it is marked as *complete*, which will move it into the archived campaigns section in Gophish. Simultaneously, based on the attachment for that campaign, the name of the attachment will be either marked as delivered or *maybe_delivered* in the project's .csv file, depending on if it is in the list of commonly blocked attachments or not.

Lastly, the functionality to archive the attachments into .iso or .zip files is also included, should the security analyst want to test how the client handles containerized versions of different groups of attachments.

5.1.2 Passive components

The passive components of the payload delivery implementation consist of a machine hosting a DNS server, email server and traffic relay. the DNS and email servers are configured with SPF, DKIM and DMARC records to create a legitimate email environment. When an email is sent with Gophish, it will go through the email server with the correct SPF flags to make the email as legitimate as any regular email. The traffic relay is used such that when the tracking pixel used by Gophish is activated, it is received on the legitimate domain and relayed to the Gophish application internally.

5.2 Payload Detonation

Implementing the payload detonation testing came with the choice between the two found methods of transmission through HTTP(S) or DNS. After experimentation with both a webserver to receive POST requests, and a Canarytokens server to receive data through DNS requests, the choice was made to use the HTTP(s) method. Both are a valid options, but due to the required encoding and formatting of data for transmission via the DNS hostname, the second option was deemed more ideal. Not every maldoc payload may support the easy transformation of data to facilitate the right format for Canarytokens, while almost all code environments have easy methods to make a web request.

The implementation of the first payload detonation testing method consists of two parts: a web-catcher and the same python program as Payload delivery, with an option to check and process the logs of the web-catcher. Naturally, the maldocs themselves are a part of the setup as well, but these are variable and the responsibility of the security analyst to create. The web-catcher we created is a simple web server that logs incoming connections and saves any post requests to a file. This file can then be inspected by our python program, which filters the data with the format as specified in Chapter 4.2 and notes that information to a file named *detonation_results.csv*. The security analyst can then inspect this file to assess which payloads managed to detonate. Additionally, any maldoc created to facilitate the method writes the data that is sent to the web server to a local file on the testing environment, such that this information can also be read manually, in the case of a failed transmission.

5.3 The program

A visualization of the components of the implementation and how they work together can be seen in Figure 6.

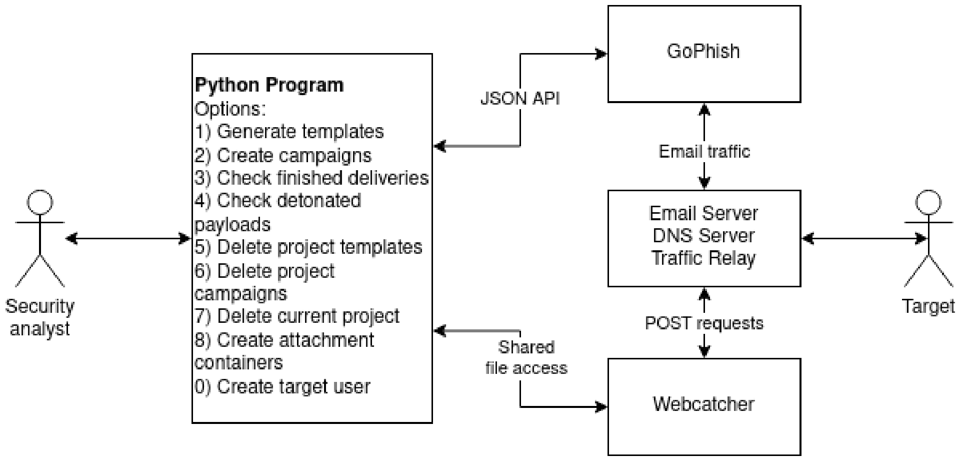


Figure 6: Schematic of the interacting components of the implementation

5.4 Test results

To verify the implementation, the program was used to perform assessments on a number of Secura's clients in a live environment using 35 file samples from different categories as described in Chapter 2.3.1. As can be seen in Appendix A.1, we notice a similar result between each client that the program was tested on. The majority of executable files such as .exe, .bat, etc. are blocked by all clients. Likewise, the .docm word macro filetype did not get through any firewall, however neither of the clients blocked word macro template files, or the macro files from excel or powerpoint, which can contain the same harmful macros that a .docm file can.

To demonstrate the payload detonation testing system, two maldoc were opened and executed on two different windows hosts. The resulting HTTP requests were caught by the webcatcher, and processed by our program. The resulting data can be seen in Appendix A.2. In these examples, the Windows antivirus did not detect that any code was executed. In other tested macro's, the antivirus did detect and stop code execution, therefore preventing any communication being established with the webcatcher.

6 Conclusion

In this study, we proposed a technique for a security analyst at a cybersecurity company to automatically test payload delivery and payload detonation in phishing campaigns. Our technique involves using a tracking pixel for payload delivery testing and including the attachment name in the email as a backup confirmation of payload arrival. The technique for payload detonation testing involves executing of a console command by the maldoc on the target system, followed by communicating the resulting output to the security analysts server through http(s) or DNS.

We implemented these methods as a program that cybersecurity company Secura can use in their phishing campaigns. The methods are easy to implement for a company interested in replicating the implementation or adapting it to fit specific requirements. However, it is important to note that our method may not be suitable for all types of campaigns and further research is needed to evaluate its effectiveness in different contexts. Secura and the clients the program was tested on were pleased with the results, especially with the value that can be provided by the payload delivery part with a low amount of effort and time. Secura may seek to offer it as a service as an addition to their phishing attachments.

Future directions for research could include refining and extending our proposed techniques to better handle a wider range of phishing campaigns, as well as exploring related research questions such as the effectiveness of different types of tracking methods or the impact of various command executions and communication methods on payload detonation testing.

References

[1] SpecterOps. Revisiting phishing simulations, 2022. https://posts.specterops.io/revisiting-phishing-simulations-94d9cd460934.

[2] J. Klensin. Simple mail transfer protocol, oct 2008. https://www.rfc-editor.org/rfc/rfc5321.

[3] M. Crispin. Internet message access protocol - version 4rev1, mar 2003. https://www.rfc-editor.org/rfc/rfc3501.

[4] M. Rose J. Myers. Post office protocol - version 3, may 1996. https://www.ietf.org/rfc/rfc1939.txt.

[5] P. Resnick. Internet message format, Apr 2001. https://www.rfc-editor.org/rfc/rfc2822.

[6] Benjamin Fabian, Benedict Bender, and Lars Weimann. E-mail tracking in online marketing: Methods, detection, and usage. 03 2015.

[7] T. Hansen, D. Crocker, and P. Hallam-Baker. Domainkeys identified mail (dkim) service overview, July 2009.

[8] S. Kitterman. Sender policy framework (spf) for authorizing use of domains in email, version 1, April 2014.

[9] M. Kucherawy and E. Zwicky. Domain-based message authentication, reporting, and conformance (dmarc), March 2015.

[10] Z. Alkhalil, C. Hewage, L. Nawaf, and I. Khan. Phishing attacks: A recent comprehensive study and a new anatomy. 3, 2021.

[11] M. Rouse. Phishing definition. https://searchsecurity.techtarget.com/definition/phishing. Accessed nov 6, 2022.

[12] M. Jakobsson and S. Myers. *Phishing and countermeasures: understanding the increasing problems of electronic identity theft.* John Wiley and Sons, New Jersey, 2006.

[13] C. Abad. The economy of phishing: A survey of the operations of the phishing market. *First Monday*, 2005.

[14] Microsoft. Blocked attachments in outlook. https://support.microsoft.com/en-us/office/blocked-attachments-in-outlook-434752e1-02d3-4e90-9124-8b81e49a8519.

[15] Google. File types blocked in gmail. https://support.google.com/mail/answer/6590?hl=en#zippy=%2Cmessages-that-have-attachments.

[16] Fazle Ullah, Mike Edwards, Roshan Ramdhany, Ruzanna Chitchyan, Muhammad Ali Babar, and Awais Rashid. Data exfiltration: A review of external attack vectors and countermeasures. *Journal of Network and Computer Applications*, 101:18–54, 2018.

[17] A Al-Bataineh and G White. Analysis and detection of malicious data exfiltration in web traffic. In *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. IEEE, 2012.

[18] S. Peneti and B.P. Rani. Data leakage prevention system with time stamp. In *Information Communication and Embedded Systems (ICICES), 2016 International Conference on*. IEEE, 2016.

[19] R Koch, M Golling, and GD Rodosek. Behavior-based intrusion detection in encrypted environments. *IEEE Communications Magazine*, 52(7):124–131, 2014.

[20] K. Berlin, D. Slater, and J. Saxe. Malicious behavior detection using windows audit logs. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*. ACM, 2015.

[21] R. Rajamenakshi and G. Padmavathi. An integrated network behavior and policy based data exfiltration detection framework. In *Proceedings of the Fifth International Conference on Fuzzy and Neuro Computing (FANCCO-2015)*. Springer, 2015.

[22] P. Mockapetris. Domain names - implementation and specification, 1987. `https://www.ietf.org/rfc/rfc1035.txt`.

[23] Canarytokens. `https://canarytokens.org/`.

[24] Gionathan Armando Reale and Benjamin Zinc Loft. Canarytokens: An old concept for a new world. *Scientific and Practical Cyber Security Journal (SPCSJ)*, 3(1):66–68, 2022.

[25] Gophish. Gophish - open-source phishing framework, 2022. `https://github.com/gophish/gophish`.

[26] Gophish. Gophish - open-source phishing framework api, 2022. `https://docs.getgophish.com/api-documentation/`.

A Appendix

A.1 Client delivery results

File Name	Client1	Client 2	Client 3
sample.exe			
sample.bat			
sample.pif			
sample.cmd			
sample.chm			
sample.wsf			
sample.vbs			
sample.lnk			
sample.msi			
sample.url		✓	
sample.hta			
sample.com			
sample.js			
sample.wsh			
sample.scr			
sample.vhd			
sample.cab			
sample.img			
sample.rtf	✓	✓	✓
sample.iso			
sample.html	✓	✓	✓
sample.dotm	✓	✓	✓
sample.ppsm	✓	✓	✓
sample.xlsm	✓	✓	✓
sample.docm			
sample.xll			
sample.ppt	✓	✓	✓
sample.xltm	✓	✓	✓
sample.xls	✓	✓	✓
sample.xlam	✓	✓	✓
sample.pptm	✓	✓	✓
sample.doc	✓	✓	✓
sample.ppam	✓	✓	
sample.txt	✓	✓	✓
sample.docx	✓	✓	✓

A.2 Detonation results

Project	File Type	File Name	Machine	Additional
host1	docm	macro1.docm	john_johnson	none
host2	xlsm	macro1.xlsm	mauk_l	none