Hardening MFA web applications to counter evolving transparent phishinging attack vectors

Philipp Liermann 20-475873-20

Supervisor: Prof. Dr. Raad Bin Tareaf

June 9, 2024

Contents

0.1 Abstract

As of today most online services have implemented multi-factor authentication to prevent malicious third-party actors from using stolen login credentials. Despite this, a new technique referred to as transparent proxy phishing still provides malicious actors the means of acquiring a victims private credentials and MFA tokens during a classical social engineering based phishing attack. The attacker is able to effectively duplicate legitimate services by using specifically configured reverse proxy servers that relay traffic from the victims browser to the original service while collecting and intercepting wanted information like passwords and credit card numbers. Also the ease with which transparent proxy phishing can be executed is alarming. Open-source toolkits and publicly available resources have made it simpler than ever for attackers to deploy such sophisticated schemes without a deep understanding of web development or cybersecurity. This elevation in threat sophistication underscores the urgent need for a corresponding advancement in defensive measures. This creates a critical security threat, as the technique can be used to gain unauthorized access to otherwise secure systems. To resolve this issue, it is imperative to develop and implement effective measures to counteract transparent phishing and alike. This paper will outline different strategies that can help organizations and developers to protect their applications against this new attack vector.

0.2 Problem statement

As an online service provider it is crucial to protect the privacy and security of your users. Due to the fact the transparent or so called reverse proxy phishing attacks are becoming more common and sophisticated implementing additional active security measures to detect them is essential. The goal of this paper is to provide a comprehensive overview of the current strategies to detect transparent phishing attacks by watching incoming client traffic. As this often requires TLS fingerprint to detect known reverse proxy software the paper will also try to propose a new approach of enumerating and actually validating supported TLS ciphers of a client.

0.3 Research question

How can we protect web applications against transparent proxy phishing attacks? Based on this question the following sub questions were derived: What exactly are classical phishing attacks in general and how do they differ from the new attack vector?
How big is the threat imposed by this new type of phishing attack?
How can a webserver know that a client connection is being relayed through a transparent phishing toolkit?

0.4 Objective

The objectives of this research are threefold. Firstly, it aims to elucidate the mechanics and impact of transparent proxy phishing, underscoring why it is a growing concern in the cybersecurity community. Secondly, the study evaluates the current state of research, identifying gaps in existing defenses against this attack vector. Finally, through rigorous experimentation and practical simulations, the paper proposes new, effective strategies to mitigate this threat, contributing valuable insights to the arsenal of cybersecurity defense mechanisms.

0.5 Theoretical foundations

Cyberfraud

is a form of internet-based fraud, usually involving the use of false identities and/or stolen information to illegally obtain money, property, or services. Cyberfraud is an increasingly pervasive problem that is becoming increasingly difficult to combat, as fraudsters become more sophisticated in their methods. In 2020, cyberfraud was estimated to cost the global economy over $6 trillion[?], with the financial sector suffering the most damages. The social, economic and reputational costs of cyberfraud can be incredibly damaging, and can range from the loss of money, to identity theft, to the disruption of businesses. Cyberfraud has become so pervasive that it is essential for businesses and individuals to take measures to protect themselves from it. This includes using strong passwords, using two-factor authentication, and staying up to date with the latest security protocols.

A Phishing attack

is a type of cyberattack in which an attacker attempts to gain confidential information, such as passwords, credit card numbers, or other sensitive information, by sending emails or other messages disguised as legitimate entities. These messages often include malicious links to faked login prompts that will steal the victims credentials upon entering them. These attacks are becoming increasingly sophisticated and difficult to recognize, making it important for everyone to remain vigilant and take steps to protect against them.

A HTTP reverse proxy

is a type of proxy server that retrieves resources on behalf of a client from one or more servers. This type of proxy is sometimes referred to as a "gateway" or "tunneling" proxy because it acts as a gateway for the traffic to and from the server. A reverse proxy will typically receive a request from a client, then forward that request to an appropriate server on the same network. It then retrieves the response from the server and sends it back to the client. This type of proxy server is most often used in enterprise networks to protect against malicious traffic, to balance load between multiple servers, and to cache static content.
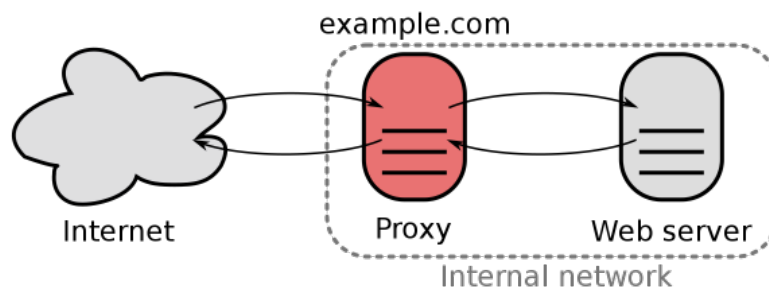

Figure 1: How a HTTP reverse proxy works

Man in the Middle (MITM) attacks

are a type of cyberattack in which an attacker intercepts and modifies communication between two parties without their knowledge. This type of attack is often used to steal sensitive information, such as login credentials, credit card numbers, or other personal information. MITM attacks can be difficult to detect, as the attacker can intercept and modify communication without the knowledge of the parties involved.
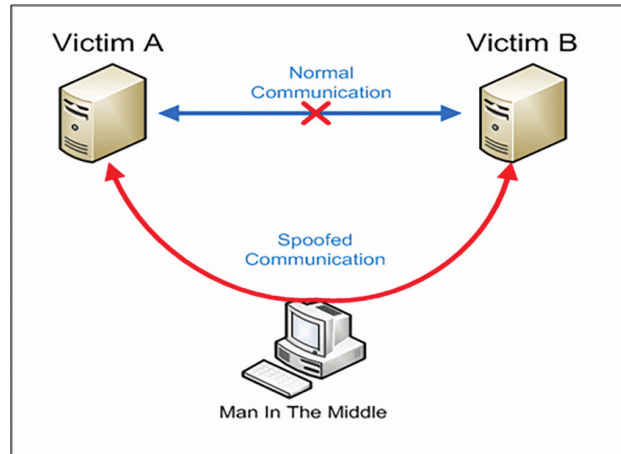

Figure 2: How a MITM attack works

Transparent proxy phishing

is a new technique used by attackers to intercept and steal multi-factor authentication (MFA) tokens from unsuspecting users. Instead of copying HTML code from the original page that the attacker is trying to impersonate, this new attack uses a HTTP reverse proxy to just redirect the users traffic to the original page. The attacking proxy operator can view and modify all traffic that is going through it while the victim sees a one by one copy of the original login page. By doing so login credentials and 2FA tokens can be extracted easily.

TLS

is a cryptographic protocol that provides end-to-end security for data sent between a client and a server. It is widely used to secure web traffic, email, and other types of data. TLS is the successor to SSL, and is often referred to as SSL/TLS. If a webserver is using TLS its URL starts with the well known https:// prefix.

TLS Fingerprinting

is a technique used to identify the TLS implementation of a client or server by analyzing the handshake process. This can be used to identify which software is used by client or server. Industry standards for fingerprinting algorithms have existed for a long time. These include: JA3, JA3N and the whole JA4+ family.

Record Type	Record Version	Record Len	Handshake Type	Message Len	Message Version
1 byte	2 bytes	2 bytes	1 byte	3 bytes	2 bytes

Random	Session ID Len	Session ID	Cipher Suites Len	Cipher Suites
32 bytes	1 byte	SID len bytes	2 bytes	CS len bytes

Compression Methods Len	Compression Methods	Extensions Len	Ext 1 Type	Ext 1 Len
1 byte	CM len bytes	2 bytes	2 bytes	2 bytes

Ext 1 Data	Ext 2 Type	Ext 2 Len	Ext 2 Data	...	Ext n Type	Ext n Len	Ext n Data
ext 1 len bytes	2 bytes	2 bytes	ext 2 len bytes	...	2 bytes	2 bytes	ext n len bytes

Figure 3: The TLS ClientHello packet that is used for fingerprinting the client

Nginx

is a popular open-source web server and reverse proxy server. It is used by millions of websites to serve web pages and other content. Nginx is known for its high performance, stability, and low resource usage. It is also known for its flexibility and extensibility, as it can be easily extended with third-party modules and plugins. Because Nginx can easily be configured to act as a reverse proxy and supports TLS, it is often used as a so called TLS terminator, meaning that it terminates the TLS connection and forwards the unencrypted traffic to a HTTP backend app.

Docker

is a platform for developing, shipping, and running applications. It allows developers to package their applications and dependencies into containers, which can then be run on any system that has Docker installed. Docker containers are lightweight, portable, and self-sufficient, making them an ideal platform for deploying applications.

HTML

is the standard markup language for creating web pages and web applications. It is used to structure and present content on the web. HTML is used in conjunction with CSS and JavaScript to create interactive and visually appealing web pages.

Unicode

is a standard for encoding, representing, and handling text in most of the world's writing systems. It is used to represent characters from all of the world's writing systems, including Latin, Cyrillic, Greek, Arabic, Hebrew, Chinese, Japanese, Korean, and many others. Unicode is used in many modern software applications, including web browsers, word processors, and operating systems. In computer science literature, Unicode symbols are often represented as U+ followed by a hexadecimal number.

7

Contents

Wireshark

is a popular open-source network protocol analyzer that is used to capture and analyze network traffic. It is widely used by network administrators, security professionals, and developers to troubleshoot network problems, analyze network traffic, and detect security vulnerabilities.

0.6 Current state of research

Some research has been done on this topic already, still we are far away from production ready plug-n-play solutions that can reliably identify transparent phishing toolkits.

0.7 Research Design

First, a thorough analysis of existing and scientific papers on transparent proxy phishing toolkits will be undertaken. This literature review will critically evaluate prior research findings, methodologies, and theoretical frameworks deployed in the study of phishing attacks and defense mechanisms. By doing so, it will identify potential gaps in the current body of knowledge and establish a solid theoretical foundation for subsequent experimental work. In parallel, experimental research will be conducted by setting up controlled attack simulations using various open-source transparent proxy phishing toolkits . These toolkits, commonly used by adversaries in real-world scenarios, will be systematically evaluated to uncover flaws and weaknesses in their attack implementations. Each experiment will involve detailed documentation of the setup, execution, and outcomes of the phishing simulations, ensuring reproduc ibility and transparency in the research process.

The experimental phase will involve the following steps:

Selection of Toolkits

A range of widely-used open-source reverse proxy phishing toolkits will be selected based on criteria such as popularity, functionality , and community support. Examples may include Evilginx, Modlishka, and Mura ena.

Simulation Setup

Virtual environments replicating real-world web services will be created to serve as targets for the phishing simulations. These environments will be configured to log detailed information on the interactions between the phishing toolkits and the target web services.

Execution of Attacks

Each selected phishing toolkit will be deployed against the simulated web services. Various attack scenarios will be tested to evaluate the robustness and effectiveness of the phishings toolkits. Specific metrics for assessment may include detection evasion, data exfiltration capabilities , and overall success rate of phishing attempts.

Data Collection and Analysis
Data generated from the attack simulations will be meticulously collected and analyzed. Statistical techniques will be employed to identify patterns, correlations, and significant findings that shed light on the strengths and weaknesses of the transparent proxy phishing toolkits.

Validation of Findings
To ensure the reliability and validity of the findings, additional experiments may be conducted with different configurations and settings, including testing the toolkits against various web security measures. Based on the insights gained through literature review and empirical experiments , this thesis will then develop and outline easy-to-follow strategies to protect web services from reverse proxy phishing attacks. These strategies will be crafted in a manner that allows for intuitive implementation by security professionals and web service administrators, aiming to enhance the overall resilience of web services against such threats.

In conclusion, this research design integrates both the theoretical and practical aspects of cybersecurity research, adopting a rigorous and systematic approach to advance current understandings of reverse proxy phishing toolkits and contribute to the development of effective defense mechanisms.

1 Literature Review

1.1 Overview of Multi-Factor Authentication

Multi-Factor Authentication (MFA) is a security system that requires more than one method of authentication from independent categories of credentials to verify the user's identity for a login or other transaction. The most common categories of things the can be used as a second or third factor are [2]:

- Something the user knows (e.g., a password or PIN)
- Something the user has (e.g., a smartphone, a hardware token, or a smart card)
- Something the user is (e.g., biometric data such as fingerprints, facial recognition, or iris scans)

MFA is used to protect the user from unauthorized access to their accounts, and is widely used in the financial and healthcare industries, as well as in government and military applications. MFA is also used in consumer applications, such as online banking and e-commerce. The use of MFA is growing rapidly, as more and more organizations recognize the need for stronger security measures to protect their users and their data.

MFA is a critical component of a strong security posture, and is an essential tool for protecting against a wide range of cyber threats, including phishing, credential theft, and identity theft. MFA is also an important tool for protecting against insider threats, as it can help to prevent unauthorized access to sensitive data and systems. MFA is also an important tool for protecting against the growing threat of cyberfraud, as it can help to prevent unauthorized access to financial accounts and other sensitive information.

1.2 Phishing Attacks: Evolution and Impact

Phishing attacks are a type of cyberattack in which an attacker attempts to gain confidential information, such as passwords, credit card numbers, or other sensitive information, by sending emails or other messages disguised as legitimate entities. These messages often include malicious links to faked login prompts that will steal the victims credentials upon entering them. These attacks are becoming increasingly sophisticated and difficult to recognize.

Phishing has evolved over time, from simple scams to sophisticated attacks that are difficult to detect. In the early days of the internet, phishing attacks were relatively

simple and easy to recognize. However, as technology has advanced, so have phishing attacks. Today, phishing attacks are often highly sophisticated and difficult to detect, making them a significant threat to individuals and organizations.

Classical phishing fake login pages were usually simple HTML copies of the original login page, but connected to a fake backend service that would store the entered credentials and redirect the user to the original page or a fake error page. This way victims could recognize that they had fallen for a fake login page and update their credentials before the attacker could use them.

The new transparent proxy phishing attack vector is a new technique used by attackers to intercept and steal multi-factor authentication (MFA) tokens from unsuspecting users. Instead of copying HTML code from the original page that the attacker is trying to impersonate, this new attack uses a HTTP reverse proxy to just redirect the users traffic to the original page. The attacking proxy operator can view and modify all traffic that is going through it while the victim sees a one by one copy of the original login page. By doing so login credentials and 2FA tokens can be extracted easily. Also the victim will not recognize that he has fallen for a phishing attack, because the website actually behaves as expected. The user will not be redirected to a fake error page or the original page after entering his credentials, because a real session is established with the original server.

In summary the advantages over the classical fake HTML login page based attack are:

- Knowing basic web development techniques including HTML, CSS and JavaScript is not required to setup a phishing attack anymore

- The victim will not recognize that he has fallen for a phishing attack

- The attacker can view and modify all traffic that is going through the proxy, including the entered credentials and 2FA tokens

- Custom HTML and JavaScript can be injected into the original page to steal even more information from the victim using social engineering techniques

In April 2017 a John Hopkins University Student named Xudong Zheng published a blog post with a proof of concept on how he used a homograph attack to impersonate apple.com. He used the Cyrillic letter "a" (U+0430) instead of the Latin letter "a" (U+0061) in the domain name.
This way he was able to register the domain "xn--80ak6aa92e.com" which looks exactly like "apple.com" in the browser address bar. This attack was possible because the browser displayed the domain name in its punycode representation, which is a way to represent Unicode with the limited character subset of ASCII used for internet host names.
The combination of this attack with a transparent proxy phishing attack is especially dangerous, because the URL in the address bar will look exactly like the original URL

and also behave exactly like it due to usage of transparent proxying. It is close to impossible even for a educated user to recognize the he is being lured into a phishing attack. Luckily the browser vendors were quick to react and implemented a fix for this issue.

By reading literature on this topic, it becomes clear that performing a transparent proxy phishing attack is becoming easier and easier. This is because of the increasing number of open source reverse proxy phishing toolkits that are available on the internet. These toolkits are designed to allow read teams and cyber security researchers to set up and run transparent proxy phishing attacks in a controlled environment. However, these toolkits can also be used by malicious actors to perform real-world attacks.

The following are some of the most popular open source reverse proxy phishing toolkits:

- Modlishka [?] is a low level HTTP reverse proxy framework that allows the attacker to intercept and modify all traffic that is going through it. It is written in Go and can be easily extended with custom plugins. Modlishka also provides a web interface to copy collected session data into an attackers browser to perform session hijacking attacks. It is also capable of injecting custom HTML and JavaScript into the original page to steal even more information from the victim.

- Evilginx2 [?] is a more advanced transparent phishing toolkit that is also written in Go. It provides same capabilities as Modlishka, but also includes ready to use templates and a custom DNS server. Its more user friendly and easier to use than Modlishka, but also less flexible and extensible.

- Murena [?] is a transparent phishing toolkit that is written in Python. It is not as advanced as Modlishka and Evilginx2, but is still capable of intercepting and modifying all traffic that is going through it. It is also capable of injecting custom HTML and JavaScript into the original page.

1.3 TLS Fingerprinting Algorithms

In general performing TLS fingerprinting on a client connection works by analyzing the TLS ClientHello message that is sent by the client to the server during the TLS handshake process. This message contains a lot of information about the client's TLS implementation, including the version of the TLS protocol that the client supports, the list of supported cipher suites and extensions.

To generate a unique fingerprint out of this information several algorithms already exist. The JA3 algorithm proposed first by Salesforce [?] collects values from the ClientHello into four different array lists and formats them into one comma separated string that
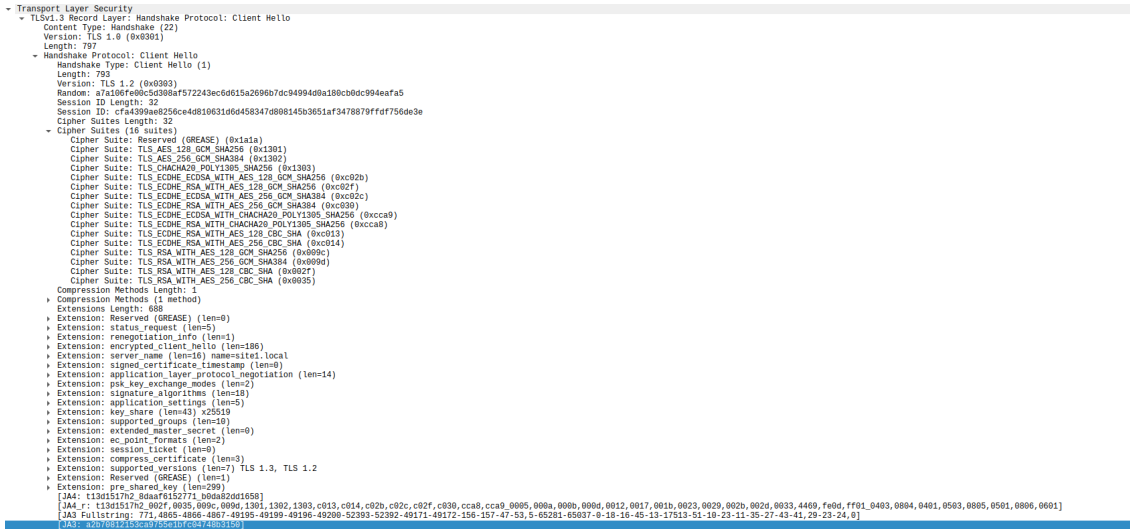
Figure 1.1: A TLS ClientHello packet captured using Wireshark

gets hashed. In detail the values used from ClientHello are: SSLVersion, Cipher, SSLExtension, EllipticCurve, EllipticCurvePointFormat.
An example derived from a ClientHello looks like this:

```
"769,47-53-5-10-49161-49162-49171-49172-50-56-19-4,0-10-11,23-24-25,0"
```

If no SSL extensions are used the field will be left empty. Example:

```
"769,4-5-10-9-100-98-3-6-19-18-99,,,"
```

Generating a MD5 hash from that string will leave us with the following 32 characters: "ada70206e40642a3e4461f35503241d5". JA3 also needs to ignore values for non existing extensions, because some TLS clients are using Google's GREASE (Generate Random Extensions And Sustain Extensibility). This is a feature proposed by Google to break wrongly implemented TLS servers. It may sound controversial at first, but comes with good intentions. In a internet draft paper by D. Benjamin [**?**] he explains that its better to break some production systems in place instead of risking flawed TLS implementations to spread and risk outages of global scale.
JA3 seems to be a good candidate for fingerprinting transparent phishing toolkits at first, but it after some testing it comes clear that JA3 can not be used to fingerprint instances of Google Chrome, due to it's reliance on the order of the cipher extension list in the ClientHello package. Google chrome uses a feature called "TLS ClientHello extension permutation" that prevents TLS fingerprinting for said reason. This can be bypassed by sorting the numeric values by size. Open source implementations of the JA3 algorithm with normalized extension orders exist under the name "JA3N". Sadly in our experimentation we were not able to reliably identify versions of Google Chrome using JA3 for yet unkown reasons.

Luckily a better alternative to the JA3 algorithm exists. The JA4+ family is a set

of new network fingerprinting algorithms developed by FoxIO [**?**]. The JA4+ family also comes with a fingerprinting algorithm for TLS called JA4. JA4 is superior to JA3 in many ways. It supports HTTP 3 including its UDP based transport protocol QUIC and normalized the order of ciphers and extensions by default. It uses a different output format than JA3 which consists of three seperatable parts that more verbose and human readable in general. JA4 provides a more reliant and expressive fingerprinting solution for this papers use case. Other open-source implementations of JA4 are avaible and include a wireshark addon and an nginx fork. The later one will be used in our lab setup to collect JA4 fingerprints.
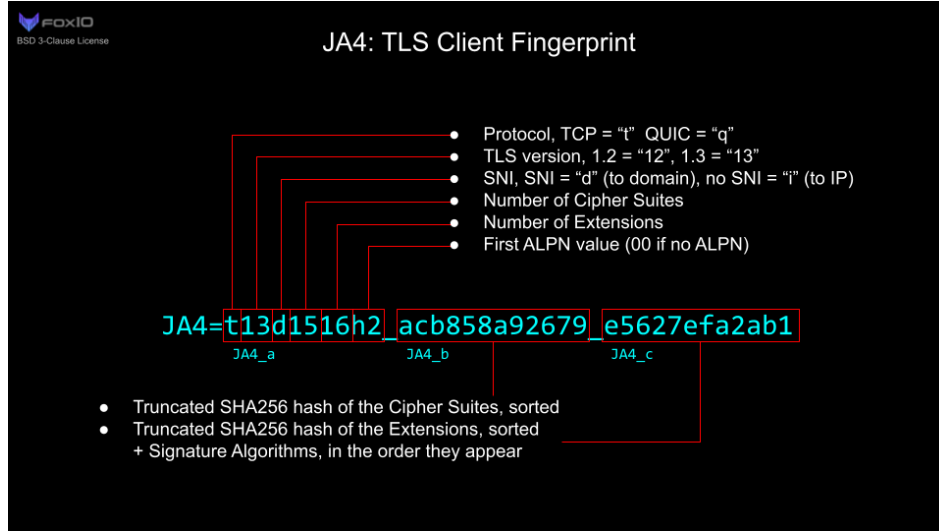

Figure 1.2: JA4 TLS client fingerprint format

TODO: the google feature is "to reduce potential ecosystem brittleness" and not for privacy. Quote more from: https://chromestatus.com/feature/5124606246518784

1.4 Existing Countermeasures

The most prominent research on this topic is conducted in the paper "Catching Transparent Phish: Analyzing and Detecting MITM Phishing Toolkits" by Brian Kondracki Et al. [1]. The team of four provides a detailed overview of the most used transparent phishing toolkits and provides a list of statistical features that can be used to detect those by testing the TLS server of the toolkit. The list of features includes TLS fingerprints, User-Agents, simulated global network delay and TCP packet timings from benchmarks of said tools. Their model was able to detect the most common open-source MITM transparent phishing toolkits with an accuracy above 99%. It is intended to be used with a script that scans the internet for phishing sites that are run using MITM phishing toolkits. Both is available as open-source on GitHub under the name PHOCA which means seal in latin.[**?**] Through this research some of the chosen features for MITM

toolkit server detection will be used to watch client traffic for signs of it being relayed through a transparent phishing toolkit.

1.4.1 Mismatch of User-Agent and TLS Fingerprint

A broader and more flexible approach then the previous blacklist solution which tries to block or flag client requests using the TLS implementation of known transparent phishing toolkits, is to focus on detecting requests from TLS reverse proxies in general. The attack vector that is facilitate by abusing reverse proxies to spy on a users TLS encrypted session is more commonly known as a TLS MITM (Man in the Middle) Attack. The American content delivery network operator Cloudflare which is also known for DDoS mitigation and other cyber security related services is running an open-source monitoring service called MALCOLM which stands for "Measuring Active Listeners, Connection Observers, and Legitimate Monitors". It provides statistics about observed HTTPS Interceptions. A HTTPS Interception defined in Cloudflare's own terms is a request that comes from either a "A device has a root certificate installed that allow an intermediary to decrypt and inspect traffic" or "An origin server provides its TLS private key to a third party (like a reverse proxy) that does TLS termination" [**?**]. The second definition means exactly the kind of TLS MITM attacks which we are trying to detect.

The monitoring platform is powered by "MITMEngine" (Monster-In-The-Middle Engine) that Cloudflare describes as their HTTPS interception detector. MITMEngine is an open-source software written in go. It works by matching TLS fingerprints of requests to known browser User-Agent strings. If a mismatch is found the request is marked as potentially crafted by HTTPS Interception. This strategy again relies on maintaining a huge database of User-Agent and TLS Fingerprint pairs. Cloudflare states on the MALCOLM dashboard website that they can only identity 60% of the clients that are served by their internal network. Especially the detection of Android based devices is not reliably possible due to the large amount of different Android based operating systems and browsers. Readers of the MALCOLM website are encouraged to contribute User-Agent TLS fingerprint pairs to the MITMEngine git repository to help overcome this issue.
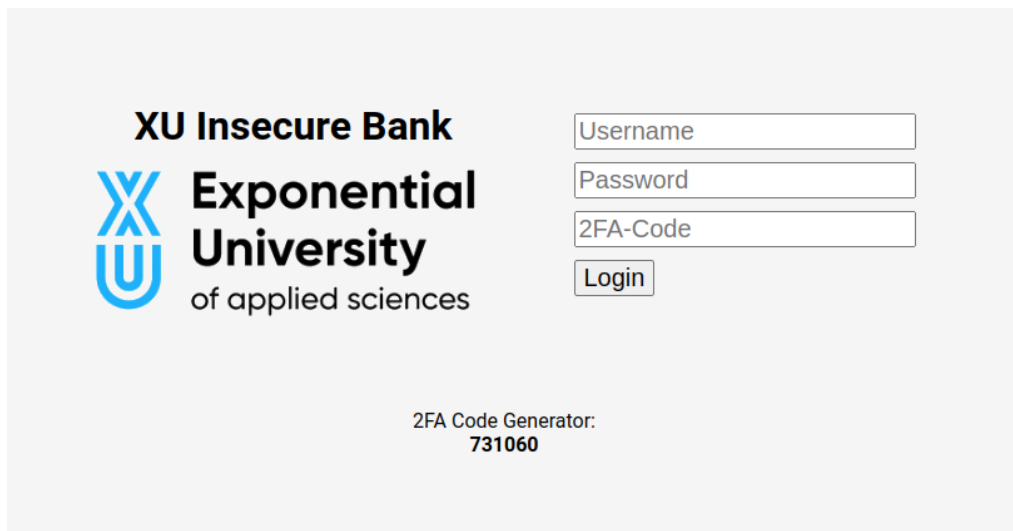
1.4.2 Round Trip Time Analysis

2 Experimentation

2.1 Simulation of Transparent Phishing Attacks

To test and analyze the capabilities of the most popular open source transparent phishing toolkits a controlled lab environment will be set up. The goal is to find flaws in their attack implementation and to develop and test new or improved countermeasures against these attacks.

First a web application will be set up that is secured with a multi-factor authentication system. Then the open source transparent phishing toolkits will be used to set up attack simulations against this web application. The local web application will be running on a local machine, but still be secured with a valid self-signed TLS certificate. Our own certificate authority will have to be installed in the browser used for testing, a recent version of google chrome in this case. In a real world scenario an attacker would also have to buy a valid domain name. For our simulated attacks an entry in the hosts fill be sufficient to redirect the traffic to the local machine.

The test environment web application implements a very basic credential based login

Figure 2.1: Web application secured with a demo two-factor authentication system

system with two-factor authentication. For demo purposes the two-factor authentication logic will accept any 6 digit number as valid token.

The frontend of this application is connected to a NodeJS based backend that is using express.js for routing and parsing of incoming requests. After one provides login credentials in the frontend a authentication endpoint in the backend will be called with the provided credentials. If the credentials are valid the backend will redirect the user to his profile page. This response will also contain a session cookie that will be used to authenticate the user in the future.
For an attacker, stealing the content of that session cookie is enough to gain full access to that account. This is also true for most real world web applications where cookie based authenticaion is used, but this alone is not a securiy issue. Nearly 99% of all websites are TLS secured today [**?**] meaning that no one except the user and server can see the content of that HTTP request including the session cookie.

In our simulated attack scenario it will be our goal to steal the content of this session cookie to overtake the victims session. In the real world an attacker is likely saving login credentials eg. email and password from all requests he intercepts with his transparent phsihing toolkit's mitm proxy. Additionally he may inject custom HTML and JavaScript into the original page to steal even more information from the victim using social engineering techniques, for example by asking for an additional multi factor authentication token that he can use in the future. For the purpose of this experiment we will only focus on stealing the session cookie.
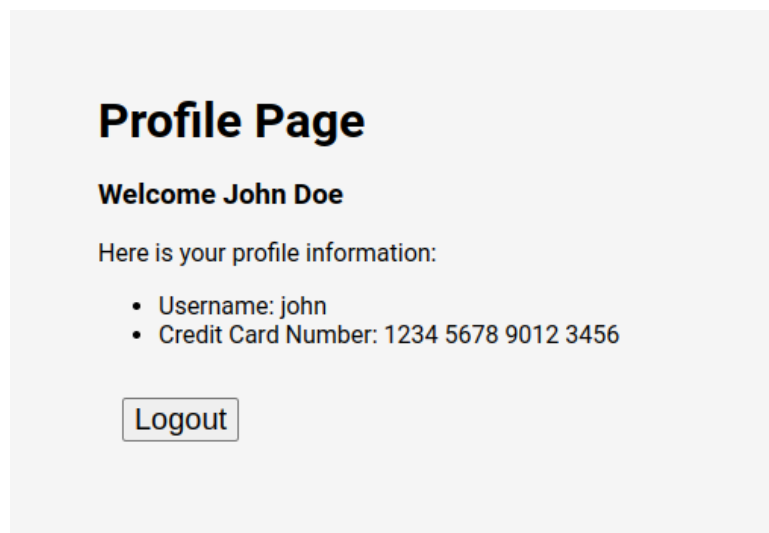
Figure 2.2: Profile page containing confidential information

2.2 Testing of Countermeasures

2.2.1 Blacklisting well known TLS Fingerprints

The first countermeasure that we are trying to implement in our web applications backend is to blacklist TLS fingerprints of well known transparent phishing toolkits. This way we can detect if a client connection is relayed through a transparent phishing toolkit and show the user a warning or ignore the whole request. This approach comes with some obvious downsides:

- The attacker can change the TLS fingerprint of his transparent phishing toolkit by modifying the TLS stack of the toolkit or by using a different toolkit with an unknown fingerprint

- The attacker can spoof the TLS fingerprint of the victim's client

- The attacker downgrades to HTTP and does not use TLS at all

Example implementation of a basic blocklist implemented using a custom nginx fork that supports JA4 fingerprinting:

```
server {
  listen 443 ssl;
  server_name xu-bank.com;

  # ... other ssl configuration

  # blocklist
  map $http_ssl_ja4 $allowed_client {
    default                                1;
    "t13d191000_9dc949149365_e7c285222651"  0; # evilginx2
  }

  location / {
    # block the request if the clients fingerprint is on the blocklist
    if($allowed_client = 0) {
      return 403;
    }

    # forward the request to the backend
    proxy_pass http://backend;
  }
}
```

The same approach can be turned into a whitelist by inverting the logic. This way only clients with a known good TLS fingerprint will be allowed to access the web application. This approach is more secure, but also comes with the downside that new clients will not be able to access the web application until their TLS fingerprint is added to the whitelist. Maintaining a trusted JA4 fingerprint database would require a lot of fingerprint collecting before said whitelist can be used in production as many different browsers and TLS implementations exist and change frequently.

2.2.2 Verify supported ciphers

A TLS ClientHello packet sent by a client always contains a list of supported ciphers. The list of ciphers and its orders can be used to fingerprint the client. Many open-source mitm revese proxy solutions are already capable of spoofing the supported cipher list in the ClientHello package. To detect client requests from said reverse proxies more reliably, verifying the implementation of each cipher could be helpful. A proof of concept cipher spoofing tool would need to support many different ciphers. To overcome this issue using a TLS server stack that supports as many ciphers as possible is required. TODO: TABLE, Talk about why we choose openssl.

OpenSSL 3.1.1 supports 159ciphers in the build that is shipped with Fedora39 Linux.

```
$ openssl ciphers -v 'ALL:eNULL'
TLS_AES_256_GCM_SHA384           TLSv1.3 Kx=any     Au=any     Enc=AESGCM(256)            Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256     TLSv1.3 Kx=any     Au=any     Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256           TLSv1.3 Kx=any     Au=any     Enc=AESGCM(128)            Mac=AEAD
TLS_AES_128_CCM_SHA256           TLSv1.3 Kx=any     Au=any     Enc=AESCCM(128)            Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384    TLSv1.2 Kx=ECDH    Au=ECDSA   Enc=AESGCM(256)            Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384      TLSv1.2 Kx=ECDH    Au=RSA     Enc=AESGCM(256)            Mac=AEAD
DHE-DSS-AES256-GCM-SHA384        TLSv1.2 Kx=DH      Au=DSS     Enc=AESGCM(256)            Mac=AEAD
DHE-RSA-AES256-GCM-SHA384        TLSv1.2 Kx=DH      Au=RSA     Enc=AESGCM(256)            Mac=AEAD
ECDHE-ECDSA-CHACHA20-POLY1305    TLSv1.2 Kx=ECDH    Au=ECDSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
...
$ openssl ciphers -v 'ALL:eNULL' | wc -l # Count the number of ciphers
159
```

OpenSSL may be more well known as a library for cryptographic functions, but it also contains with a command line interface. The openssl binary comes with a built-in TLS and HTTP server implementation called s_server. It can be used to quickly setup TLS servers for testing and debugging purposes. The list of supported ciphers for a server can be configured using the -ciphers option for TLSv1.2 and below and the -ciphersuites option for TLSv1.3 and above. For the sake of this paper a proof of concept solution was implemented using a simple bash script to spawn various instances of OpenSSL's s_server with different cipher configurations. To validate if a client actually supports his claimed ciphers, each instance of s_server will be configured to only support one cipher. The client will then be forced to use this cipher to establish a connection. If the client is not able to establish a connection an error will be written to a log file that is supervised by our bash script.

```bash
#!/bin/bash
# Cleanup from previous runs
killall openssl && rm -rf ./logs/*

# List of default ciphers supported by openssl s_server
ciphers=$(cat <<-END
TLSv1.3     :TLS_AES_128_GCM_SHA256
TLSv1.3     :TLS_AES_256_GCM_SHA384
...
END)

# Spawn static https server to serve check.html and api
www_port=8443
openssl s_server -key key.pem -cert cert.pem -accept $www_port -WWW 2>&1 &

# Start of test s_server port range
port=8000

while IFS= read -r line; do
cs=$(echo $line | sed 's/.*://')
echo "Spawning server with cipher: $cs"

cipher_param=""

# Detect if its TLSv1.3 or lower
is_tls13=$(echo $line | grep -c "TLSv1.3")

# Use the correct openssl command based on TLS version
if [ $is_tls13 -eq 1 ]; then
  cipher_param="-ciphersuites $cs -no_tls1_2 -no_tls_1_1 -no_tls_1 -no_ssl3"
else
  cipher_param="-cipher $cs -no_tls1_3"
fi

# Spawn s_server instance and redirect output log to file
openssl s_server -key key.pem -cert cert.pem -accept $port -www \
$cipher_param -debug >> "logs/$port-$cs.txt" 2>&1 &

port=$((port+1))
done <<< "$ciphers"

# Check all log files for errors and print overview of server status
...
```

To make a browser connect to all the created TLS servers and show connection errors verbosely creating a bunch of iframes is enough. To simplify the process of creating the iframes a simple JavaScript script can be used.

```javascript
let framesDone = 0;
const onFrameDone = () => {
  framesDone++;
  if(framesDone === SERVER_AMOUNT){
    fetch(`https://${DOMAIN}:8444/`); // Provide browser information to bash scrip
  }
}
const frames = document.getElementById('frames');
for(let i = 0; i < SERVER_AMOUNT; i++){
  const port = 8000+i;
  const url = `https://${DOMAIN}:${port}`;

  const frame = document.createElement('iframe');
  frame.src = url;
  frame.onload = onFrameDone;
  frame.onerror = onFrameDone;
  frames.appendChild(frame);
}
```

After all iframes have either loaded successfully or failed the script performs a final fetch request to another s_server that is running with the brief option. This server will collect the clients UserAgent and supported TLS ciphers. After that, before the bash script exits, it will write a result json file that contains information about each tested cipher including logs of each s_server instance that was used for cipher validation.
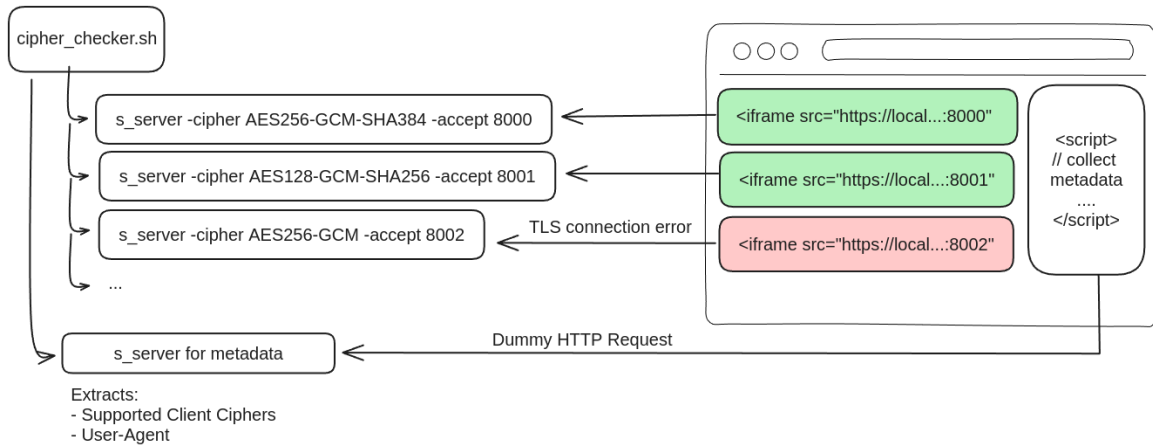

Figure 2.3: Using iframes to connect to all spawned s_server instances

To verify that this approach is working as expected across different versions of browser and TLS implementations a testing setup needed to be created. As running different versions of browsers on the same host is not easily possible, docker containers and some shell scripts were utilized.
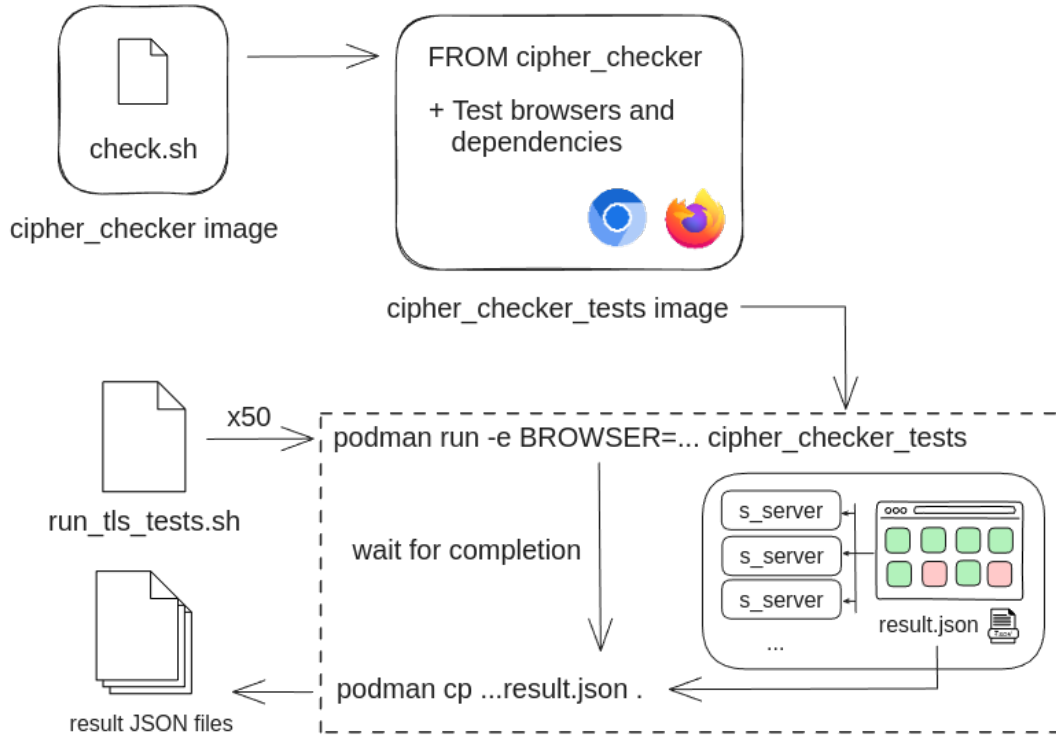

Figure 2.4: The bash script spawns multiple instances of the cipher_checker_tests which includes check.sh and a browser for testing

To parse the result json files of each browser test a python script was made. It parses each json file, compares the supported ciphers with the list of ciphers that the client told the server it supports and also visualizes the results in a HTML table.

Figure 2.5: The python script compares the sent client ciphers from the ClientHello with the s_server connections that were successful

	TLS_AES_128_GCM_SHA256	TLS_AES_256_GCM_SHA384	TLS_CHACHA20_POLY1305_SHA256	ECDHE-ECDSA-AES128-GCM-SHA256	ECDHE-RSA-AES128-GCM-SHA256	ECDHE-ECDSA-AES256-GCM-SHA384	ECDHE-RSA-AES256-GCM-SHA384	ECDHE-ECDSA-CHACHA20-POLY1305	ECDHE-RSA-CHACHA20-POLY1305	AES128-GCM-SHA256	AES256-GCM-SHA384
Chromium 59.0.3071.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 60.0.3112.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 61.0.3163.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 62.0.3202.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 63.0.3239.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 64.0.3282.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 65.0.3325.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 66.0.3359.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 67.0.3396.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 68.0.3440.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 69.0.3497.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 70.0.3538.0	error	error	error	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 71.0.3578.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 72.0.3626.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 73.0.3683.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 74.0.3729.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 75.0.3770.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 76.0.3809.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 77.0.3865.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 78.0.3904.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 79.0.3945.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 80.0.3987.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 81.0.4044.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 86.0.4240.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 87.0.4280.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 88.0.4324.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 89.0.4389.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 90.0.4430.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 91.0.4472.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 92.0.4515.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 93.0.4577.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 94.0.4606.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
Chromium 95.0.4638.0	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok

Figure 2.6: Cutoff results table screenshot of 56 different versions of the Chromium browser

2.2.3 Combined Score Based Approach

3 Results

3.1 Findings from Simulations

Present the data collected from the simulations, providing analysis and interpretation.

3.2 Effectiveness of Current Defenses

Assess the effectiveness of existing defenses against reverse proxy phishing based on your findings.

3.3 Proposed Solutions

Introduce any new solutions or improvements to existing solutions developed through your research.

4 Discussion

4.1 Implications of Findings

Discuss the broader implications of your findings for cybersecurity practices and MFA implementation.

4.2 Limitations and Future Research

Acknowledge any limitations of your study and propose areas for future research.

5 Bibliography

[1] Brian Kondracki, Babak Amin Azad, Oleksii Starov, and Nick Nikiforakis. Catching transparent phish: Analyzing and detecting mitm phishing toolkits. 2021.

[2] Joseph Williamson and Kevin Curran. Best practice in multi-factor authentication. *Semiconductor Science and Information Devices*, 3, 05 2021.