

Analyzing the Resilience of Two-Factor Authentication Techniques against Runtime Phishing Attacks

Master Thesis

Author(s):

Kellenberger, Remo

Publication date:

2024

Permanent link:

<https://doi.org/10.3929/ethz-b-000670938>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Analyzing the Resilience of Two-Factor Authentication Techniques against Runtime Phishing Attacks

Master's Thesis

Remo Kellenberger
Department of Computer Science

Advisor: Daniele Lain
Supervisor: Prof. Dr. Srdjan Čapkun

April 29, 2024

Abstract

Despite their inherent security weaknesses, username and password combinations remain the most popular form of authentication. Their weaknesses stem from factors such as password database leaks and users reusing their passwords across multiple services. To mitigate these risks, almost every online service offers two-factor authentication (2FA) to provide an additional layer of security. However, most 2FA schemes remain vulnerable to runtime phishing attacks. In runtime phishing, an attacker can bypass 2FA by immediately using stolen credentials on the legitimate platform and relaying additional verification to the user.

In this thesis, we propose Playphish, a modular phishing framework capable of simulating runtime phishing attacks. Using Playphish, we analyzed the defensive mechanisms of eleven high-traffic service providers. A user study evaluated the framework's performance against real user accounts. The results highlight a lack of resistance against runtime phishing attacks. The initial data collection stage revealed that even without sophisticated techniques, attacks were successful against nine of the eleven services. The evaluation on real user accounts yielded similar results. Phishing attacks were successful for all user accounts for six of the remaining nine services. The only resistance to the phishing attempts was the use of CAPTCHAs.

This study highlights the challenges in defending against runtime phishing attacks. Despite the availability of phishing-resistant alternatives such as FIDO authentication, most services still permit the use of weaker alternatives. This underscores the need for improved security practices to combat runtime phishing attacks.

Acknowledgements

Firstly, I would like to thank my advisor Daniele Lain for his continuous support and expertise. The regular meetings helped me stay on the right path and provided valuable insights.

Special thanks go to Prof Dr Srdjan Čapkun, who gave me the opportunity to write my thesis in the Systems Security Group and provided me with the infrastructure and resources needed for this thesis.

I would also like to thank my family and friends for their endless support over the past six months. A special thank you goes to my colleague Colin for the numerous discussions over lunch and for testing the experimental platform. Thank you, Selina, for proofreading my thesis and keeping me motivated during tough times. Thanks to my flatmates for challenging my ideas and socializing during long hours in the home office.

Finally, I would like to thank all participants in my user study for their time, effort, and trust. Without you trusting me with all your credentials, this thesis would not have been possible.

Contents

1	Introduction	1
1.1	Runtime Phishing	1
1.2	Research Goals	2
1.3	Contributions	3
1.4	Thesis Organization	3
2	Background	5
2.1	Phishing	5
2.1.1	Mitigation of Phishing Attacks	6
2.2	Two-Factor Authentication (2FA)	7
2.3	Runtime Phishing	8
2.3.1	Runtime Phishing Attack	8
2.3.2	Mitigation of Runtime Phishing Attacks	9
2.4	Risk-based Authentication	10
2.4.1	CAPTCHA	10
2.5	FIDO	10
2.5.1	WebAuthn API	11
2.5.2	Client to Authenticator Protocol (CTAP)	11
2.5.3	Limitations	11
3	Problem Statement	13
3.1	Research Questions	13
3.2	Methodology	13
3.3	Evaluation	14
3.3.1	Threat Model and Scope	14
3.3.2	Study Design	14
4	Playphish	15
4.1	Overview	15
4.2	Architectural Design	16
4.3	Target	16
4.3.1	Step Configuration	17
4.4	Implementation	18
4.4.1	Playphish	19
4.4.2	TargetPage	19
5	Experimental Platform	21
5.1	Overview	21
5.2	Frontend Javascript Application	22
5.2.1	Authentication	23
5.3	Python RESTful API	23

5.4	PostgreSQL Database	23
5.4.1	Participant	24
5.4.2	Participant Info	24
5.4.3	Target	24
5.4.4	Attempt	24
5.5	Python WebSocket API	24
5.6	Deployment	24
5.7	Security Considerations	24
6	Data Collection	27
6.1	Overview	27
6.2	Experimental Setup	27
6.2.1	Targets	27
6.2.2	Browser	27
6.2.3	Test Accounts	28
6.3	Results	29
6.3.1	Google	29
6.3.2	Coinbase	29
6.3.3	Binance	29
7	Evaluation	31
7.1	Overview	31
7.2	Scope and Attacker Model	31
7.3	Study design	31
7.3.1	Participants	32
7.3.2	Browser	32
7.3.3	Proxy	33
7.4	Results	33
7.4.1	CAPTCHAs	33
7.4.2	Notifications	34
7.4.3	Other Resistance	34
8	Discussion	37
8.1	Ethics and Security	38
8.1.1	Data Protection	38
8.1.2	Risks and Countermeasure	38
9	Conclusion	41
A	Account questionnaire	43

Chapter 1

Introduction

Today, our entire lives are online, so logging in to online services has become a daily routine. We may start our day by checking the latest posts on LinkedIn or Instagram and reading our emails on Gmail. The train ticket to get to the office can be conveniently purchased online. At work, we collaborate on documents using Microsoft Office directly in the browser. Upon returning home, we discover that a parcel containing our new shoes has arrived. Of course, we already knew this because we were able to track the parcel through the Post website. The invoice for the new shoes can be paid directly through e-banking.

Throughout the day, users sign in to platforms on numerous occasions, typically utilizing a combination of a username and password. Despite the inherent security weaknesses associated with passwords, they remain one of the most predominant forms of authentication. These weaknesses arise from various factors, including the common practice of users reusing their passwords for different services [55]. A study conducted by security experts in 2023 revealed that only a quarter of individuals employ strong and unique passwords [20]. Another factor is their vulnerability to phishing attacks. Phishing is a form of cybercrime in which attackers use social engineering to steal users' credentials. According to reports from the Anti-Phishing Working Group (APWG), the number of phishing attacks has been on the rise, reaching a new high in 2023. The most recent report has shown that social media (43%) and SAAS/Webmail (15%) are the most targeted industries [14].

To address the security weaknesses of passwords, organizations are promoting the use of two-factor authentication (2FA) as an additional layer of security. In the presence of 2FA, a user will be required to present a second factor in addition to their username and password. The most common types of 2FA schemes involve entering a verification code sent to the user via SMS or email, or verifying the authentication request on the smartphone through push notifications [45]. For added security, a one-time password (OTP) could also be generated directly on the user's smartphone via an authenticator app [36]. Although Two-Factor Authentication (2FA) provides clear security advantages, it remains susceptible to sophisticated phishing attacks. One type of attack that is effective against 2FA is known as runtime phishing.

1.1 Runtime Phishing

Runtime phishing is an attack that allows an attacker to bypass common 2FA schemes. Unlike traditional phishing, where stolen credentials are stored for later use, in runtime phishing, the attacker immediately uses stolen credentials to log into the legitimate website. Should the legitimate website prompt for a second authentication factor, the attacker relays this request to the user. Consequently, the attacker is capable of harvesting the credentials and second factor from the user and validating them in real time. Fig. 1.1 illustrates the flow of a runtime phishing attack. Firstly, a user is lured into visiting a malicious website that mimics a legitimate website, also known as a phishing site. The phishing site prompts the user for their login credentials. As soon as the user submits their credentials to the phishing site, the attacker opens a browser window to the legitimate website and enters the user's credentials. The

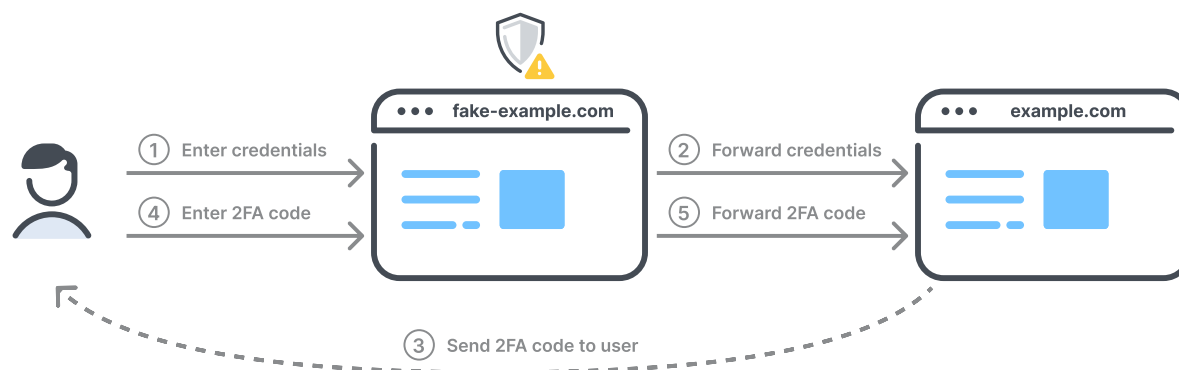


Figure 1.1: Example flow of a runtime phishing attack.

legitimate website then sends the user a message containing the 2FA code. The user, still unaware of the fraud, submits the 2FA code to the phishing site. The attacker forwards the code to the legitimate website and gains access to the user’s account.

Defenses. Runtime phishing attacks exploit a vulnerability in two-factor authentication schemes. The majority of 2FA schemes cannot verify whether the user is interacting with a legitimate website or a phishing site. Consequently, researchers have developed novel 2FA schemes to address this issue. Ulqinaku et al. [52] proposed 2FA-PP, a smartphone-based scheme that measures the round trip time between the user’s smartphone and the legitimate website. This can be used to detect an attacker who proxies requests between the user and the legitimate website. Sun et al. [48] proposed a 2FA scheme that requires the user to take a photo of the browser window and upload it to a web application. The web application then verifies that the URL matches the legitimate server.

The industry’s answer to runtime phishing is FIDO [16]. FIDO authentication is designed to address the shortcomings of conventional password-based authentication systems. Rather than relying on passwords, FIDO employs cryptographic key pairs. During the registration process, a new public/private key pair is generated on the user’s device, with the public key being transmitted to the service provider. Upon authentication, the user is required to demonstrate possession of the private key by signing a challenge issued by the service provider. The private key is never transferred from the user’s device and is only accessible to the service provider, which renders it resistant to phishing attacks. With the latest improvements, also known as FIDO2 [19], users are no longer required to use physical security keys to store FIDO credentials. Instead, they may opt to store them on their smartphones or in keychains.

However, it is unclear, if and how much runtime phishing protection is used in practice. Large service providers are usually very tight-lipped about the security mechanisms they use.

1.2 Research Goals

The goal of this thesis is to gain insights into the defensive mechanisms implemented by high-traffic service providers against runtime phishing attacks. The deployment of phishing-resistant authentication methods, such as FIDO2, is far from being complete and most providers that currently support these methods remain vulnerable to downgrade attacks [51]. These attacks involve downgrading of phishing-resistant authentication methods to weaker alternatives through social engineering. Moreover, the deployment of runtime phishing attacks is relatively straightforward, facilitated by the availability of open-source tools such as Evilginx [13] or Modlishka [10]. Consequently, these attacks are accessible to a large community of criminal actors.

Consequently, our objective is to investigate the extent to which service providers care about the protection of their users against these attacks. In addition, the techniques they use to protect their users will be analyzed. Furthermore, the study will examine whether user behavioral patterns influence the

success rate of phishing toolkits and the effectiveness of protection measures.

1.3 Contributions

We contribute Playphish, a modular phishing framework, capable of executing runtime phishing attacks. This framework leverages the browser automation library Playwright [8] to forward user credentials in real time. In the initial data collection stage, a series of runtime phishing attacks were simulated against eleven high-traffic service providers using Playphish. The development of attackers of varying sophistication levels enabled an in-depth analysis of the defensive mechanisms employed by the service providers. The least sophisticated attacker (i.e. the default configuration of the browser automation tool Playwright and no user behavior) was able to successfully execute attacks against six of the eleven services. Limiting Playphish to only use Firefox enabled the successful attack of nine services.

The second contribution is a user study to evaluate the performance of the Playphish phishing framework against real user accounts. In the user study, participants completed the runtime phishing simulation on our study website. Real user accounts with varying "internal risk scores" were used to validate the results from the assessment stage. The aim was to simulate attacks from different networks and browsers, factors that may influence the service's decision to allow or block the attack. The results demonstrated a similar pattern to that observed in the data collection phase. Service providers have not implemented significant resistance measures and have primarily relied on CAPTCHAs for protection. In the case of attacks originating from a trusted network, such as the ETH network or a home network, CAPTCHAs were not displayed for the majority of services.

In many cases, the attacks have triggered a form of risk-based authentication. This involves requesting additional verification from participants or notifying them about a suspicious login. Notifications typically include information about the platform and browser used for the suspicious login. However, all services extracted this information from the user agent string transmitted in the HTTP header of the login request. Therefore, we were able to manipulate this information for all services except Google and Microsoft.

1.4 Thesis Organization

This thesis is organized as follows. In Chapter 2, we provide a comprehensive overview of the background related to this topic. This chapter aims to establish the required foundation for subsequent discussions within this thesis. Chapter 3 defines the scope of this thesis. We provide a high-level overview as well as an implementation-level description of the Playphish phishing framework in Chapter 4 and of the experimental platform in Chapter 5. In the following chapter, we present the results of the evaluation of the framework. Chapter 6 outlines the experimental setup and provides a brief overview of the findings from the data collection stage. Chapter 7 details the user study. Finally, in Chapter 8, we discuss the results of the data collection stage and user study in the context of the research questions and conclude our thesis in Chapter 9.

Chapter 2

Background

2.1 Phishing

Phishing is a form of social engineering cybercrime wherein an attacker tricks a victim into revealing sensitive information, typically including credentials (like username and password), credit card numbers, and bank account information. Usually, the attacker communicates a socially engineered message to the victims to lure them into visiting a fake but real-looking website, also known as a phishing site. A phishing site is designed to visually mimic a legitimate platform and prompt victims to input their sensitive information. This sensitive information is then unknowingly disclosed to the attacker. Note that the definition of phishing is not consistent in the literature. Some researchers describe phishing much broader as an attack to lure victims into performing certain actions for the attacker's benefit through socially engineered messages [25]. This also involves attack scenarios, where a victim unintentionally installs malicious software after clicking on a link in a message. Sensitive data is thus extracted through the malicious software that either affects the browser or the whole computer.

Definition. For the remainder of this thesis, we refer to phishing as a social engineering attack that lures a user into visiting a deceptive website (known as phishing site) to obtain sensitive information from the user. Typically, a phishing site is designed to mimic a legitimate website.

As of 2024, phishing is still one of the major forms of cybercrime. The Internet Crime Complaint Center (IC3) of the FBI has published in their annual report that phishing is the top digital crime type by far [5]. According to reports of the Anti-Phishing Working Group (APWG), the number of phishing attacks is steadily increasing with its peak in 2023 (see Fig. 2.1). The most recent report has shown that social media (43%) and SAAS/Webmail (15%) are the most targeted industries [14].

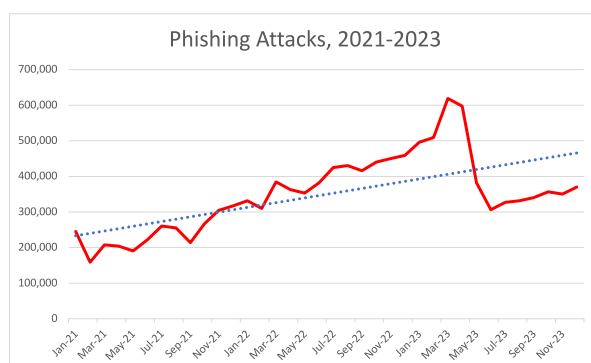


Figure 2.1: Number of unique phishing campaigns between 2021 and 2023 (Source: APWG Phishing Activity Trends Report Q4 2023 [14]).

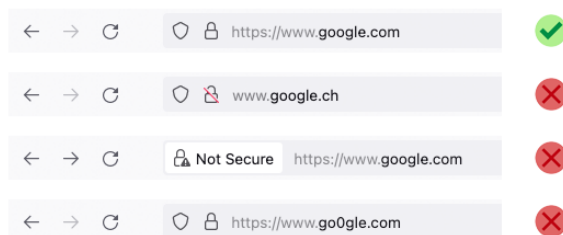


Figure 2.2: Firefox’s approach to alert users in its address bar about possible impersonations of Google.

2.1.1 Mitigation of Phishing Attacks

Although phishing remains one of the biggest threats, there have been many initiatives to protect users from this attack. This section gives an overview of some of the most important anti-phishing techniques.

Phishing Detection

Phishing sites can perfectly resemble legitimate sites in appearance, however their weak spot lies in their domain names. An attacker attempting to impersonate a legitimate website is unable to register an identical domain name to that of the authentic website with a valid SSL certificate. Fig. 2.2 illustrates an example of how Firefox alerts users when encountering attempts to impersonate Google. In the first case, the website hosted at `www.google.ch` lacks a valid certificate, as indicated by a crossed-out lock symbol. If the attacker installs a certificate issued to a domain name other than the one displayed, a “Not secure” banner is presented. In the third case, the attacker replaced the character ‘o’ with the number ‘0’. This is an example of typosquatting, where the attacker attempts to make the domain name appear legitimate. The domain name, in conjunction with the browser’s alerting system, is an effective method for detecting phishing attempts. However, it does not prevent users from accessing the phishing page if they are unable to recognize the security alerts.

One of the easiest ways to protect users from phishing is to maintain collections of known phishing domains, commonly referred to as blacklisting and whitelisting. Blacklist filtering mechanisms are used in all major browsers, such as Google Chrome featuring Google Safe Browsing [32]. However, blacklisting is only effective against known deceptive sites and is useless against sites that have not yet been recorded. Various researchers tried to overcome this limitation by using machine learning to detect malicious domain names [31, 44]. Whitelisting, on the other hand, is an approach to keep track of trusted sites and mark all other sites as suspicious. Unfortunately, it is almost impossible to maintain a complete list of trusted sites.

The aforementioned strategies have solely relied on the domain name to identify phishing sites. However, as mentioned earlier, phishing sites are designed to mimic legitimate websites. Therefore, it is also possible to identify phishing sites by measuring their similarity to legitimate websites. This can be achieved by comparing textual elements, such as HTML, CSS, or the Document Object Model (DOM) [58, 37]. The DOM represents the structure of a website, organized as a logical tree. Comparing textual elements is not always successful. Attackers may replace text with images, add invisible content, or obfuscate the source code. Thus, other methods that use computer vision to compare the visual similarity of phishing sites have been developed [23]. More advanced strategies even employ deep learning for comparison [1]. Similarity-based detection relies heavily on the dataset of legitimate websites, which is used for comparison. Ideally, as many legitimate websites as possible should be included to provide an accurate assessment, although this can result in large datasets and consequently slow comparisons.

Education

We have seen various strategies to detect phishing campaigns, but even the most sophisticated methods have limitations and fail to achieve perfect accuracy. Therefore, educating users about phishing is a popular strategy to fill this gap. Researchers have shown that there exist multiple successful ways to approach phishing education. This could be in the form of an interactive online game [46], embedding training into the daily business of users [28, 29] or more traditional face-to-face education [47].

In the previous sections, we saw that despite numerous attempts to protect users against phishing, there is no completely secure method. Recent studies in organizations have demonstrated, that although phishing training and awareness have high success rates, it is unlikely that an organization can eliminate phishing threats [30]. The following section will outline a strategy to minimize the harm of traditional phishing methods.

2.2 Two-Factor Authentication (2FA)

Passwords remain the most commonly used authentication method, despite their inherent security weaknesses. People often use weak passwords that are easy to guess based on features of their everyday lives such as their name, birthday, hobbies, relatives, or friends. As a result, human-generated passwords tend to lack randomness [54] and are vulnerable to guessing attacks. Password reuse is another bad practice, wherein people use the same or slightly modified versions of their passwords across different services [55]. If a user's password gets leaked in a data breach, it can lead to the compromise of their other accounts. Research has shown that neural networks can be used to compromise more than 16% of user accounts in less than 1000 attempts if one of their passwords has been leaked [40]. Nevertheless, even if all passwords are strong and users refrain from reusing the same password across different services, attackers may still successfully obtain credentials through phishing.

Two-factor authentication has been developed to address the weaknesses of password-based authentication and provide an additional layer of security. Instead of relying solely on a password, users are required to present two types of proof of their identity [42]:

- (1) Something they *know*: This could be a password, a PIN code, or a security question
- (2) Something they *have*: This could be a smartphone, laptop, or a security key
- (3) Something they *are*: This could be a fingerprint or other biometrics

Sometimes, literature uses the term multi-factor authentication (MFA) instead of two-factor authentication (2FA). MFA, in contrast to 2FA, refers to the scenario where a user is required to present *two or more* of the above authentication types. For the rest of this thesis, we will use MFA and 2FA interchangeably.

Various methods of two-factor authentication exist. Typically, the second factor involves the use of SMS or email-based verification codes that the user receives after submitting their password. However, for added security, users may also generate a one-time password (OTP) directly on their smartphone through an authenticator app (e.g. Google Authenticator [36]). There are also more convenient methods available that do not require users to manually enter a code. Instead, users can accept the login directly on their smartphone via push notifications [45]. In the academic community, we have also seen novel 2FA schemes that leverage the ambient sound [24, 57] as a convenient second factor.

However, all these 2FA schemes have a common vulnerability. They are susceptible to runtime phishing attacks, where an attacker immediately uses phished credentials, including second factors, to log in to the authentic website. In the next few sections, we will provide a detailed description of such attacks and present methods that attempt to mitigate these attacks.

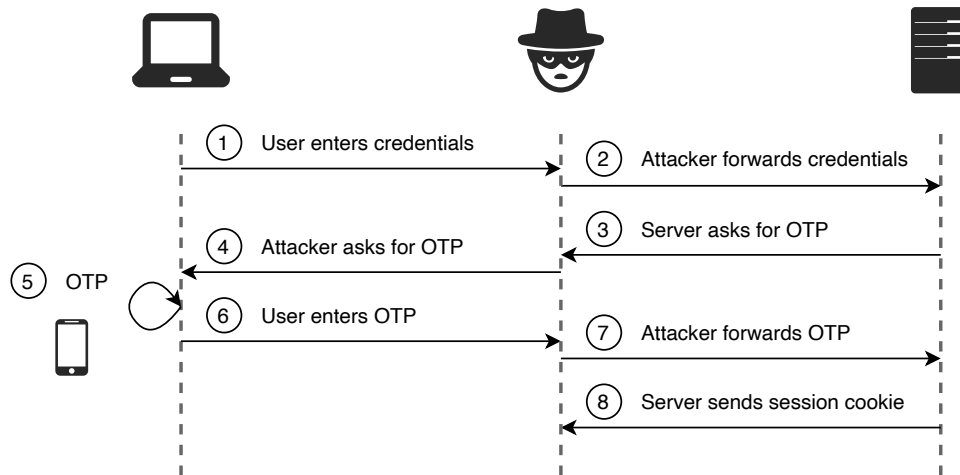


Figure 2.3: Example flow of a runtime phishing attack with OTP 2FA.

2.3 Runtime Phishing

The widespread adoption of two-factor authentication significantly decreased the risk posed by traditional phishing methods. The sole possession of a username and password is no longer sufficient to gain access to a victim’s account. This set the ground for more sophisticated attacks such as runtime phishing. Runtime phishing is an interactive phishing attack in which the attacker intercepts and forwards all communication between the legitimate website and the victim. This type of attack is also referred to as an Adversary-in-the-Middle (AiTM) attack. Two security researchers recently demonstrated the use of runtime phishing to access Tesla accounts, ultimately granting them full control over the car [38].

Before delving into the details of such attacks, it is important to first understand the fundamentals of user authentication on websites.

User Authentication. A website typically maintains the state of an authenticated user, so that the user does not need to re-enter their credentials when navigating between pages. However, the underlying protocol, Hypertext Transfer Protocol (HTTP), is stateless by design. To bridge the gap between stateful websites and the stateless HTTP protocol, a user identifier needs to be included in each HTTP request. The common method to achieve this is by using session cookies. Upon each login, the server creates a cookie associated with the user, enabling them to access their account and perform certain actions. However, this convenience comes at a price: an attacker possessing this cookie can also gain access to the account and can perform the same actions.

2.3.1 Runtime Phishing Attack

Fig. 2.3 illustrates a simple runtime phishing attack. The laptop icon represents the user being phished, the person wearing a mask represents the attacker, and the server icon represents the legitimate website. The user is initially presented with a page that mimics the legitimate website and requests their username and password. Once the user has submitted their credentials, the attacker immediately uses the stolen credentials to log into the legitimate website. If the legitimate website prompts the attacker for a one-time password (OTP), the attacker relays this request to the user. The user, still unaware of the fraud, enters their valid OTP, which the attacker then forwards to the legitimate website. Subsequently, the attacker gains access to the user’s account by obtaining a session cookie.

In certain situations, the credential forwarding process may be carried out by a human operator. This operator would have access to a dashboard that displays the victim’s inputs. Furthermore, the operator could prompt the victim for the second factor by clicking a button. At the same time, the operator would have an open browser window with the legitimate website, where they would enter the user’s credentials.

	Reverse Proxy	Active	GitHub Stars
Evilginx	✓	✓	9,850
Modlishka	✓	✓	4,661
CredSniper	✗	✗	1,291
Muraena	✓	✓	850
ReelPhish	✗	✗	498

Table 2.1: Comparison of open source phishing toolkits as of April 2024.

In practice, the operator is more likely to be a small script rather than a human, facilitated by phishing toolkits, such as CredSniper [11] and ReelPhish [6]. Navigating the legitimate website is accomplished through the use of browser automation tools, such as Selenium [7], Puppeteer [33], or Playwright [8].

Reverse Proxy Phishing

A reverse proxy is a server that sits in front of one or multiple web servers and intercepts traffic from clients. This type of proxy server is typically used for tasks such as load balancing, caching and SSL encryption. However, the same technique can also be used for malicious purposes such as phishing.

In a reverse proxy phishing scenario, the attacker configures a reverse proxy to redirect each incoming request to the legitimate server and then relays the legitimate server’s response to the victim, intercepting all credentials and session cookies. Unlike traditional phishing methods, the attacker doesn’t need to host their own HTML templates to create a replica of the legitimate site.

The security research community has come up with fully-fledged phishing toolkits for the purpose of reverse proxy phishing, such as Evilginx [13], Muraena [49], or Modlishka [10]. Ironically, these toolkits, initially developed by security researchers, are now being misused for malicious purposes. In a year-long study conducted in 2021, scientists utilized advanced fingerprinting technologies to uncover 1220 phishing websites linked to one of these three toolkits [26]. An overview of the most popular phishing toolkits is provided in Table 2.1.

2.3.2 Mitigation of Runtime Phishing Attacks

The reason why runtime phishing attacks work is that all 2FA schemes share a common vulnerability. They cannot verify whether the user is interacting with a legitimate page or a phishing page. This section provides an overview of possible mitigation strategies for runtime phishing attacks.

The academic community came up with novel two-factor authentication schemes to address the issues of previous 2FA schemes. Ulqinaku et al. [52] proposed 2FA-PP, a two-factor authentication scheme that utilizes the Web Bluetooth API and implements a challenge-response protocol to verify the proximity between the user’s smartphone and the legitimate website. In 2FA-PP, the legitimate server initiates authentication by sending an encrypted challenge (i.e. encrypted and obfuscated JavaScript code) to the client. The challenge-response protocol starts with the smartphone transmitting the decryption key for decrypting the challenge to the client via Bluetooth. Subsequently, the client executes the JavaScript challenge that produces a response including the URL of the website. If the round-trip time of the challenge-response operation exceeds a predefined threshold, indicating that the user was interacting with a phishing page and decryption was relayed to the legitimate page, the authentication process is aborted. In addition, if the URL included in the response does not match the legitimate website, the authentication process will also be aborted.

Sun et al. [48] proposed another 2FA scheme aimed at verifying that the user interacts with a legitimate website. When the server requests additional verification, the user is prompted to capture a photo of their browser window and upload it to a dedicated PhotoAuth web application with their smartphone. The PhotoAuth web application uses Optical Character Recognition (OCR) to extract and verify the URL from the photo, instead of relying solely on the user to verify the correct URL. The

PhotoAuth link is sent to the user's smartphone through an alternative communication channel, such as a push notification, SMS, or email.

There is limited evidence outside of academia regarding the implementation of protection mechanisms against runtime phishing attacks. The emphasis is primarily on detecting malicious links before users click on them [32] or assisting users in identifying phishing websites (see Fig. 2.2). Additionally, companies are working on blocking automation platforms that attackers use to carry out runtime phishing attacks [35]. Two other approaches to mitigate runtime phishing attacks are risk-based authentication and FIDO authentication. The following sections provide an overview of these techniques.

2.4 Risk-based Authentication

Risk-based Authentication (RBA) was introduced as a dynamic approach to increase security [9]. Unlike traditional static authentication, RBA leverages contextual factors to assess the risk level associated with a particular authentication attempt. RBA monitors and collects factors such as IP geolocation and browser telemetry data, including user agent, timezone, and language, for each authentication attempt. More sophisticated implementations of RBA also consider user behavior patterns [50]. Based on the collected data, a risk score is calculated, representing the likelihood of the authentication attempt being fraudulent. Typically, this score is then divided into three categories.

1. Authentication attempts categorized as *low risk* are granted without further actions.
2. Those falling into the *moderate risk* category often require additional verification steps. This could be solving a CAPTCHA, providing a second authentication factor (e.g. email or SMS verification), or being alerted about a potentially suspicious login.
3. Attempts categorized as *high risk* are usually blocked outright.

2.4.1 CAPTCHA

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is, as the name suggests, a security mechanism designed to distinguish humans from computers [53]. Traditional CAPTCHAs require users to input text presented as a blurred or distorted image, making it difficult for computer programs to decipher. More recent versions of CAPTCHAs require users to solve image recognition tasks, such as selecting all tiles that contain a specific object (e.g. cars, pedestrian crossings, buses) or rotating an object to the correct rotation.

2.5 FIDO

The FIDO Alliance [16], an open industry association driven by hundreds of global tech leaders, developed a set of specifications to provide phishing-resistant, privacy-protecting authentication. The main objective of the FIDO Alliance is to eliminate text-based passwords and their associated attack vectors (e.g. password guessing [3], credential stuffing, or phishing). Passwordless authentication methods have been around for a long time, however, they have been used rarely except in high-security settings [4]. The reason for this was that this form of authentication required special hardware and the deployment was costly. With its latest set of specifications known as FIDO2, the FIDO alliance found a way to overcome these limitations. FIDO2 includes two specifications: W3C Web Authentication (WebAuthn) [22] and Client to Authenticator Protocols (CTAP) [17]. Instead of using text-based passwords, FIDO2 is built on public key cryptography. Passwords are replaced by cryptographic key pairs denoted as passkeys [18]. This eliminates some major disadvantages of passwords

- Passkeys are *strong and unique* by design.
- There is *no shared secret* between the user and the server. Only public keys need to be stored on the server and sensitive credentials never leave the user's device.

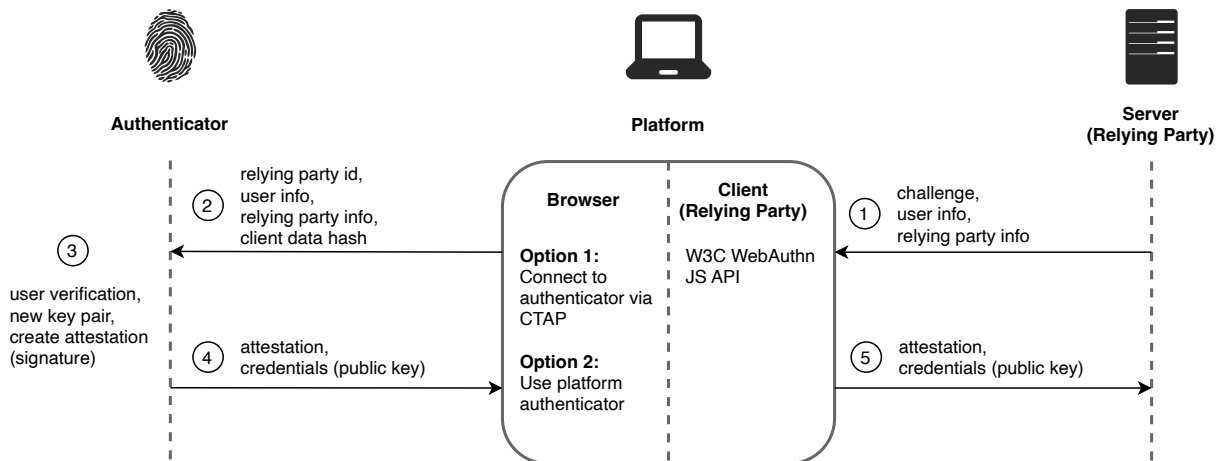


Figure 2.4: FIDO2 Registration Flow. In this example, the relying party is a web application.

2.5.1 WebAuthn API

The Web Authentication API (WebAuthn) [22] specification defines the standard web API that allows web applications to use and create FIDO credentials. At the time of this writing, it is integrated into all major browsers. The specification defines two distinct scenarios, registration and authentication. See Fig. 2.4 for an overview of the registration flow in a web application. If the user visits a website (e.g. example.com) and initiates registration, the relying party (e.g. example.com) creates a unique, random challenge. The client JavaScript implementation of the relying party then calls `navigator.credentials.create()` with the challenge and some additional input. This results in the browser prompting the user to connect to an authenticator. If the user gives consent, the authenticator receives a hash of the challenge and additional information. The authenticator then verifies the user through an authorization gesture (e.g. entering a pin or password, supplying a biometric), generates new credentials (public/private key pair), and creates an attestation based on the hash (cryptographic signature). It is important to note that the generated credentials are scoped to the relying party via the relying party identifier. This scoping is jointly enforced by the authenticator and the platform's browser. Finally, the public key and the attestation object are passed to the server to complete the registration.

2.5.2 Client to Authenticator Protocol (CTAP)

The FIDO2 specification distinguishes between two different types of authenticators, *platform authenticators* (the authenticator built into the user's device) and *roaming authenticators* (security keys over USB, BLE, or NFC). The Client to Authenticator Protocol (CTAP) [17] allows the platform (e.g. Browser, PC) to connect to roaming authenticators.

2.5.3 Limitations

FIDO represents a great alternative to traditional password-based authentication methods offering higher security. Numerous organizations have successfully adopted FIDO authentication. For instance, Google internally implemented FIDO as a second authentication factor, requiring all employees to utilize physical security keys. One year later, Google reported that there has not been a confirmed or reported account takeover since the implementation of security keys [27]. Adoption of FIDO in consumer space has lagged, despite the users' willingness to replace text-based passwords with a stronger alternative [12]. Roaming authenticators require users to carry a hardware device, and platform authenticators require re-enrolment for each new device. Moreover, recovering from device loss is very complicated. Many websites that support FIDO authentication address this problem by allowing weaker alternatives. A study conducted in 2021 revealed that all websites in Alexa's top 100 that supported FIDO were sus-

ceptible to a social engineering downgrade attack [51].

Consequently, the FIDO Alliance is working on a new version of the CTAP specification to address these issues. Its key improvements include:

- (1) **Smartphone authenticators.** Users can use their smartphone as a roaming authenticator eliminating the need for specialized hardware. While many 2FA schemes already use smartphones (i.e. OTP, SMS passcode or push notification), they remain vulnerable to runtime phishing attacks. FIDO mitigates this risk by employing Bluetooth communication between the device, where the user wants to authenticate, and the smartphone. This ensures physical proximity, which means that remote phishing attempts are prevented.
- (2) **Multi-device credentials.** The specification expects authenticator vendors to support multi-device credentials. This means, that credentials are no longer coupled to a device and in case of switching to a new device, the credentials are immediately available on the new device. The functionality is similar to password managers.

Chapter 3

Problem Statement

Passwords are a common form of authentication, but they have inherent security weaknesses. To address these weaknesses, service providers have implemented two-factor authentication (2FA) to add an extra layer of security. Unfortunately, this has led to the development of more sophisticated phishing attacks that are capable of bypassing 2FA. One such attack is runtime phishing. It is unclear, whether and to what extent service providers protect their users from this novel attack. This thesis aims to answer this question. In this chapter, we first define the research questions to be addressed, followed by an overview of the methodology and evaluation.

3.1 Research Questions

Q1: How much do service providers care about runtime phishing? The threat posed by runtime phishing is great. Multiple ways to prevent traditional phishing have been employed by service providers and as of today, almost every website offers two-factor authentication. But we want to understand if service providers also care about runtime phishing attacks and how much.

Q2: What techniques do service providers implement to protect their users from runtime phishing? We want to understand what techniques are used to protect against such attacks in the wild. We want to test, if services can detect and block authentication attempts originating from a runtime phishing attack. Furthermore, we want to understand if user behavior (i.e. mouse movement, keyboard typing, or scrolling) and habits (i.e. location, device) influence the success rate of runtime phishing.

Q3: How much does it cost to implement Runtime phishing attacks? By implementing an actual runtime phishing attack against popular services, we want to get an understanding of how expensive such attacks are. We further want to explore the scalability of these attacks.

3.2 Methodology

We developed a phishing framework capable of executing runtime phishing attacks. Existing runtime phishing toolkits, such as Evilginx, mainly rely on reverse proxies to intercept and forward all communication between the legitimate server and the victim. This comes with limitations. Initial tests have shown that certain service providers employ hidden JavaScript code to verify the website's URL, therefore protecting against reverse proxies. This code is highly obfuscated and individual to each service provider, making it difficult to reverse engineer. Moreover, these tools primarily focus on extracting session cookies. However, in situations where sessions are short-lived (e.g. online banking, password managers, cryptocurrency exchange platforms), collecting session cookies is ineffective.

Therefore, we opted for the development of a custom toolkit called Playphish that uses Playwright to control a browser. The framework was designed to be able to simulate attackers of different levels of

sophistication. By simulating a range of attacker behaviors, we aimed to determine the level of human-like behavior required to avoid detection by service providers. Furthermore, a key objective was to simplify the integration of new phishing targets to maximize scalability. This task involved developing a mechanism that enables the addition of new targets with minimal effort. The proposed solution suggests creating a small configuration file to add new targets.

During the data collection stage, we tested the phishing framework against eleven high-traffic service providers from various industries. To do this, we created new test accounts using an artificial identity. All accounts were fortified by two-factor authentication using an authenticator app or SMS-based codes.

Our approach involved simulating runtime phishing attacks against these services using the accounts, starting with the most basic attack and gradually increasing its sophistication. Playwright was instantiated with default options, and no human-like behavior or attempts to hide the browser automation library from the service providers were implemented. The attack's sophistication was gradually increased for each service until it succeeded. This entailed incorporating more human-like behavior, making it more difficult for the services to identify the automation tool and distinguish our framework from a genuine user. All services that showed some form of resistance were further analyzed. This provided valuable insights into the protection mechanisms employed by service providers.

3.3 Evaluation

To ascertain the efficacy of our proposed phishing framework in the context of a realistic scenario, we have conducted a user study by simulating runtime phishing attacks on genuine user accounts.

3.3.1 Threat Model and Scope

The assumed attacker can lure users into visiting a phishing site (e.g. through social engineering) to gain access to the user's account. However, the attacker cannot compromise the user's system, meaning they cannot install malicious software on the user's computer or smartphone to directly extract credentials, 2FA codes, or active login sessions.

Note that the focus of this thesis lies on evaluating the technical capabilities of service providers to prevent phishing attacks, rather than examining users' ability to identify such attempts. It is assumed that the user has already followed a link to a phishing site, completely unaware of the fraud. Accordingly, the user complies with all prompts and requests presented to him during the login process.

3.3.2 Study Design

The study aimed to confirm the results and findings obtained from tests conducted with test accounts during the data collection stage, using real accounts. To ensure proper validation of the initial findings, we tested our phishing framework with multiple accounts of different ages and with varying "internal risk scores". We aimed to simulate the attacks from different networks and browsers – all factors that may potentially influence the service provider's decision to allow or reject the login from our phishing framework.

For this purpose, we developed an experimental platform using the Playphish phishing framework in the backend to simulate runtime phishing attacks against the eleven high-traffic service providers. We recruited participants of different demographics from our environment. One of the main differences between a genuine user and our phishing framework is the location from which the authentication request originates. To understand if there is a correlation between a successful attempt and the origin of the attack, we routed the attacks through three different proxies. A detailed overview of the conducted user study can be found in Chapter 7.

Chapter 4

Playphish

This chapter focuses on the Playphish phishing framework, a key component of this thesis. The first part provides a high-level overview, while the second part offers an implementation-level description of the framework.

4.1 Overview

The Playphish phishing framework is a crucial component of this thesis, used in both the data collection stage and as part of the backend in our experimental platform for the user study. While the concept of developing a phishing framework capable of bypassing two-factor authentication is not novel, several security researchers have created open-source tools to achieve this. One such tool is Evilginx, which is based on a reverse proxy. With this approach, it is easy to set up a runtime phishing attack, even for those without web development knowledge, as there is no need to create a replica of the legitimate website. However, there are inherent limitations to this approach. In our initial tests with these tools, we have found that it is relatively straightforward to protect against them. Furthermore, these tools are primarily designed to extract session cookies. In cases where sessions are short-lived, such as in online banking or password managers, this approach is ineffective.

Therefore we aimed to develop a custom runtime phishing framework. We set our design goals for the framework as follows:

- **Flexibility:** The addition of a new service as a phishing target to the framework should not necessitate code changes. Instead, it should be possible to add a new service by writing a configuration file that incorporates all details specific to this service.
- **Performance:** Performance is crucial for a runtime phishing framework. When a victim submits their credentials to a phishing site, the framework needs to pass this request to the legitimate site and return the response to the victim. If this takes too long, the victim may become suspicious and question the legitimacy of the phishing site.
- **Compatibility:** A multitude of two-factor authentication (2FA) schemes exist, including SMS codes, push notifications, and one-time passwords (OTPs). Furthermore, some services employ risk-based authentication, which necessitates additional verification beyond 2FA. Ideally, the framework can accommodate all scenarios.
- **Modularity:** The framework should be capable of integration with all common frontend technologies. The generation of HTML templates that serve as replicas of authentic websites is not within the scope of this work.

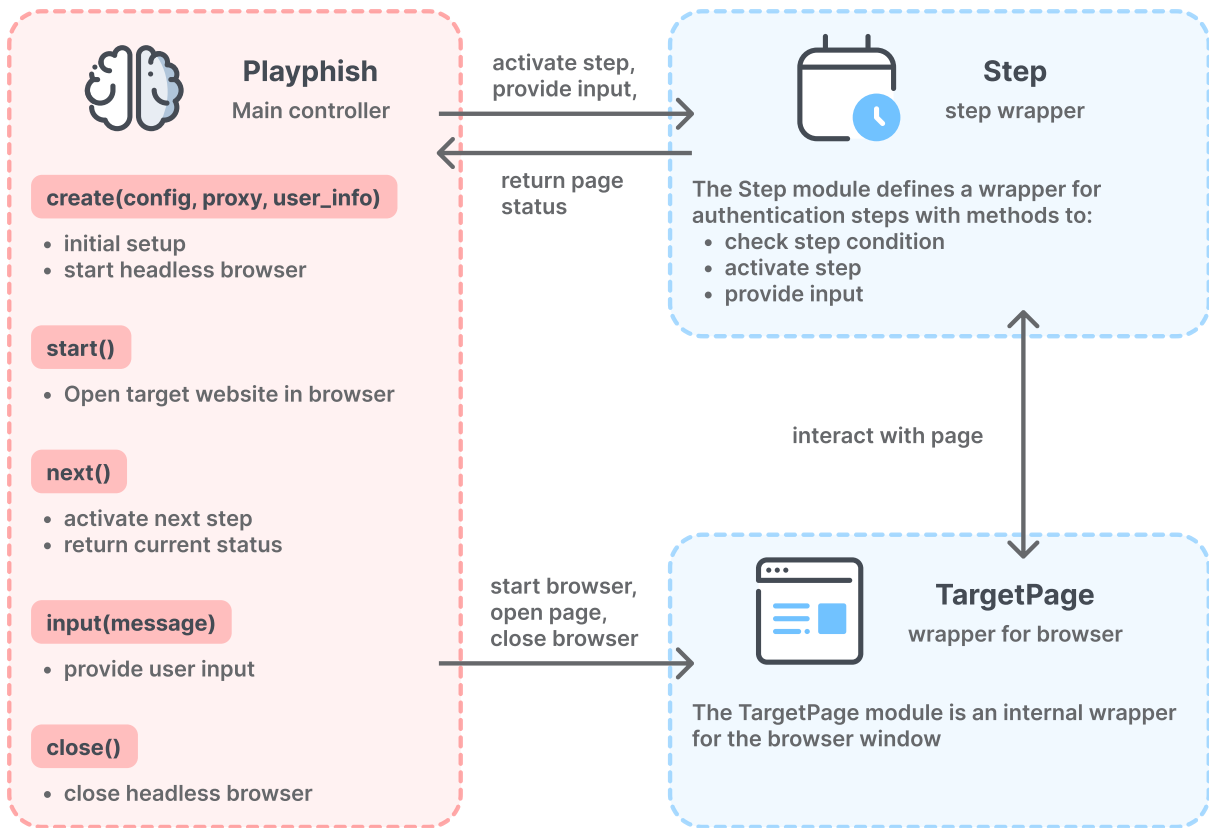


Figure 4.1: Overview of the Playphish architecture.

4.2 Architectural Design

This section provides an overview of the architectural design of the Playphish phishing framework, as illustrated in Fig. 4.1. Playphish is comprised of three main components. The entry point of the framework is the Playphish module, which serves as the main controller of the framework. Upon startup, the module receives the configuration of the phishing target, initiates a browser session with Playwright, and opens the sign-in page of the legitimate website. This is accomplished through the TargetPage module. This module represents the browser with the legitimate site as a Python object. The TargetPage offers methods for reading information from the legitimate website, inputting data to the site, and checking if the session is captured. The third component is the Step module. As will be demonstrated subsequently, a runtime phishing target can be represented as a state-transition diagram. The states in question correspond to the steps of the login procedure, which may include a username step, a password step, or a two-factor authentication (2FA) step. Examples of transitions include the victim entering the username, an error occurring on the legitimate site, or the victim accepting a push notification on their smartphone. Each such state is represented by a Python Step object. The Playphish main controller maintains a list of steps, which are then activated according to their transition logic when an event occurs.

4.3 Target

With the term target, we refer to the entity that is the target of a phishing attack, which may be a service provider such as Google, Microsoft, or LinkedIn. One of the design goals in developing Playphish is to ensure that new phishing targets do not require code changes. Consequently, a new target can be configured through a YAML file. This file consists of two sections: a general section and a steps section. The steps section models the various stages of a multi-step authentication process. The general section includes the URL of the sign-in page as well as some other metadata about the target, such as the name.

The configuration allows the specification of a list of session cookie names. Playphish will then perform a check after each event to ascertain whether the cookies are present in the browser, thereby indicating a successful phishing attempt.

The `fake_user_agent` option is used to configure Playphish, indicating whether the user agent string of the browser can be spoofed. Upon signing in to an online account from a new device, users often receive an email message notifying them of the login. This message may also include information about the device that logged in, such as the operating system and the browser. Our data collection revealed that the majority of websites parse this information from the user agent string. Consequently, by spoofing the user agent string, the victim sees the correct operating system and browser in the notification. A complete list of all options for a target is shown in Table 4.1.

	Datatype	Description
<code>name</code>	<code>str</code>	Unique identifier
<code>description</code>	<code>str</code>	Description
<code>domain</code>	<code>url</code>	Domain of the login page
<code>fake_user_agent</code>	<code>bool</code>	Fake user agent supported
<code>active</code>	<code>bool</code>	Whether the target is active
<code>supported_browsers</code>	<code>list[str]</code>	List of supported browsers
<code>chrome_options</code>	<code>dict</code>	Additional Chrome options
<code>firefox_options</code>	<code>dict</code>	Additional Firefox options
<code>safari_options</code>	<code>dict</code>	Additional Safari options
<code>context</code>	<code>dict</code>	Additional browser context
<code>auth_tokens</code>	<code>list[AuthToken]</code>	List of session cookies
<code>steps</code>	<code>list[Step]</code>	A list of steps (See Section 4.3.1)

Table 4.1: Target configuration options.

4.3.1 Step Configuration

The core of each Target configuration is the steps section. This defines, how the phishing toolkit interacts with the legitimate website and the phishing site. As previously mentioned, the sign-in flow of a website can be represented as a state-transition diagram. Fig. 4.2 illustrates the simplified state-transition diagram of the Google authentication flow.

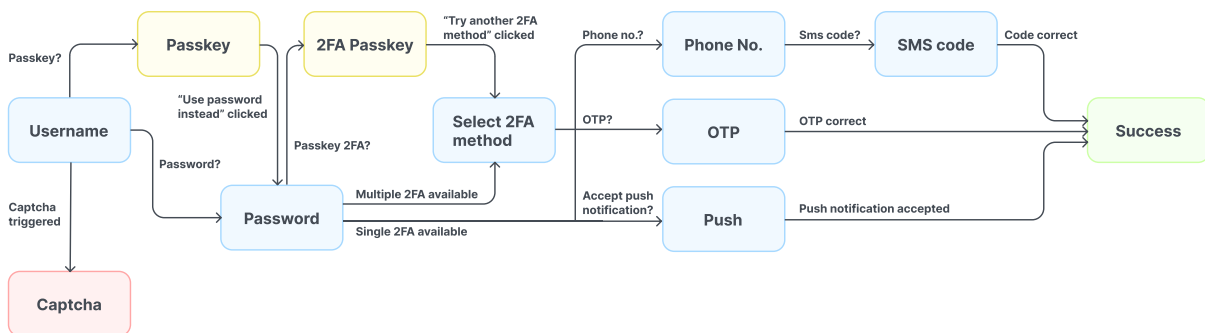


Figure 4.2: Simplified state-transition diagram of the Google sign-in flow. In Playphish, the yellow steps are run automatically in the background.

Upon accessing the Google sign-in page, the user is initially prompted to enter their username. This is followed by a request to enter their password or present a FIDO passkey. One of the functions of Playphish is that, upon being requested by the legitimate site to present the passkey, Playphish automatically bypasses this step by selecting the option to use a password instead. Subsequently, the victim is

	Datatype	Description
name	str	Unique identifier
title	str	Title of step
description	str	Description
variables	dict[str, Locator]	List of variables, which can be used in description
before	list[Action]	Action, which is executed before activation
step_locator	Locator	Locator, which defines when a step should become active
input_fields	list[InputField]	List of required inputs
next_steps	list[str]	Possible next steps

Table 4.2: Step configuration options.

presented with a password field, and the process continues as usual. This represents a form of downgrade attack, which is possible since services such as Google permit the use of weaker alternatives to FIDO passkeys.

The objective of developing Playphish was to create a tool that could be used in a wide range of login scenarios and with a wide range of authentication steps. There should be support for both those steps that require input from the victim and those that do not. The transition from one step to the next can be triggered by entering input on the phishing page or events on the legitimate website. This led to the development of the following step configuration. Table 4.2 provides a complete overview of all available options for configuring an authentication step in Playphish. The most important options are as follows:

- `step_locator`: Transitions between steps are initiated when an element on the legitimate site becomes visible. To illustrate, the appearance of the password field on the page will result in a transition from the username step to the password step. The `step_locator` option enables the specification of a DOM element whose visibility indicates when a step should become active.
- `before`: Some steps require the execution of a specific action upon activation. The `before` option allows the specification of this action. As an example, a cookie banner may prevent interaction with the page. Consequently, the Playphish framework should initially click the "accept cookies" button, thereby enabling the username field to become actionable.
- `input_fields`: The majority of steps necessitate input from the victim. The `input_fields` option enables the specification of a list of input values prompted to the victim for each step. Each entry comprises four elements: the type, name, label, and locator object. The locator object defines the corresponding input field in the DOM of the legitimate site.
- `variables`: In some instances, push notification-based two-factor authentication (2FA) necessitates that users input a numerical value that corresponds to a number displayed in the browser to validate the authentication request. Consequently, attackers must forward this information to the victim. In Playphish Steps, this can be achieved through the `variables` option, which can be incorporated into the title or description of the step.

4.4 Implementation

Playphish was developed as a Python module. The framework can be controlled via a command-line interface (CLI) or used in the backend of a phishing web application. In the data collection phase, the framework was controlled via the CLI. For the user study, it was integrated into the backend of the experimental platform. The decision to develop the framework as a Python module aligns with the design goals set forth. Consequently, the framework can be utilized with all major web frameworks,

including Django and Flask. For the experimental platform, the framework was used in conjunction with FastAPI and a Next.js frontend.

4.4.1 Playphish

The Playphish module serves as the primary controller within the framework. Upon the initiation of a phishing attack, an object of the Playphish class is instantiated via the `create` method. The method accepts the target configuration and some optional arguments, such as the browser or proxy to be used during the attack. There is also an option to pass the user agent string of the victim. Upon the creation of the Playphish object, the browser session is initiated in the background, and the legitimate site is opened via a call to the `start` method. Calling the `next` method, the module returns a message indicating the current status and the information about the current step (see Table 4.2). If the current step requires input, a phishing site could prompt the victim for the input, and the data can be passed to the Playphish framework through the `input` method. A simple example of how to use Playphish is presented in Listing 1 below.

```

1  async with async_playwright() as p:
2      # Create Playphish object with a target configuration and start the browser
3      target = await Playphish.create(
4          p,
5          config,
6      )
7      await target.start()
8      while True:
9          # Get the current status of the authentication flow
10         message = await target.next()
11         if message.type in ["input_required", "input_error"]:
12             # Get the input from the victim (e.g. through a phishing site or
13             # command line)
14             formData = {}
15             await target.input(
16                 MessageIn(type="submit", formData)
17             )
18         elif message.type in ["success", "error", "unsuccessful"]:
19             break

```

Listing 1: Simple example of using the Playphish module.

4.4.2 TargetPage

All interactions with the automation-controlled browser are conducted via the TargetPage Python module. In a runtime phishing scenario, the time elapsed between a victim submitting their credentials to the phishing site and receipt of a response from the phishing site is of significant importance. If that interval is too long, the victim might become suspicious and question the legitimacy of the phishing site. This led to the decision to use Playwright [8] to control the browser. Playwright stands out from its competitors, particularly regarding its exceptional performance. The Playwright locators were a fundamental element in the development of Playphish. Their functionality enabled the framework to identify and interact with specific elements on the target page, including the option to click, fill in, or observe them. The TargetPage module offers a wrapper around this concept, making it accessible to other modules within the Playphish framework.

Chapter 5

Experimental Platform

This chapter focuses on the description and implementation details of the experimental platform. The experiment platform was used to evaluate our phishing framework described in Chapter 4 and answer the research questions detailed in Chapter 3.

5.1 Overview

The experimental platform is a web application that allows participants of the user study to interact. It includes a landing page with a brief introduction to the topic of runtime phishing and a description of the study's scope. From there, participants can proceed to the consent page, where they can view and agree to the detailed consent form by checking the box and submitting the form. A new participant is created in the database and a JSON Web Token (JWT) is generated with their information. This token is then used to authenticate the participant for further requests in the backend. Participants with a valid JWT token are automatically directed to the dashboard page. The dashboard page shows a table with the eleven services against which the runtime phishing attacks are simulated in the experiment. Clicking on a service initiates the experiment. Participants are directed to a questionnaire about their accounts. After submitting the questionnaire, the actual runtime phishing simulation begins. The participant is presented with a login form that is connected through a WebSocket with the Playphish phishing framework in the backend. The content of the login form is dynamically altered based on the messages received through the WebSocket and the participant is guided through the authentication process. Once the phishing attempt is complete, the user will be presented with a message and given the option to return to the dashboard. The results of the phishing attempt will be stored in the database and the content of the

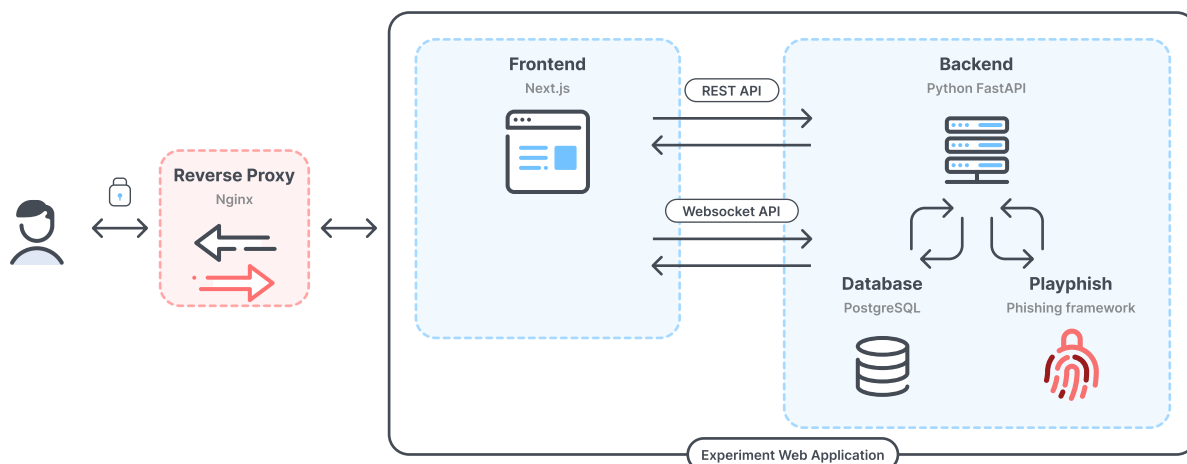
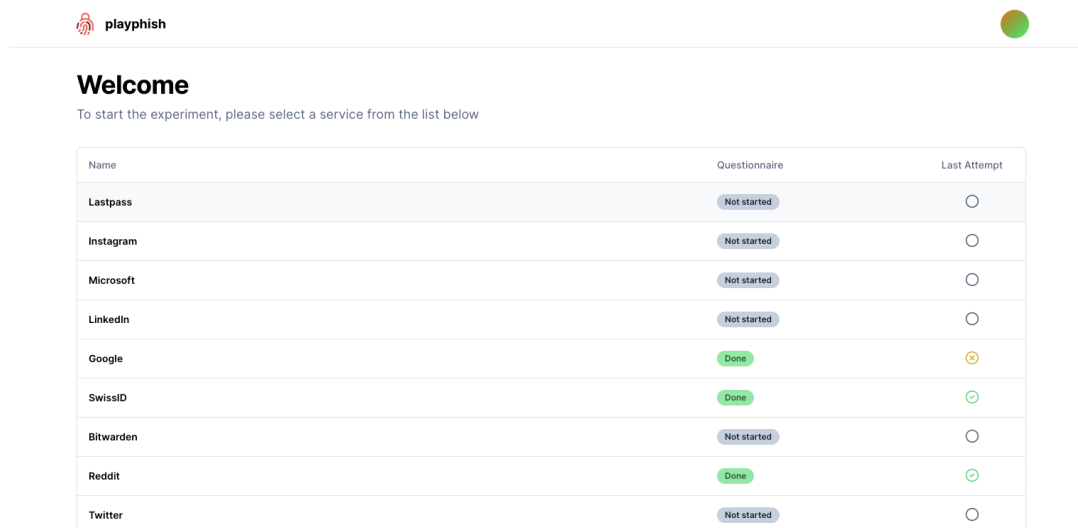


Figure 5.1: System architecture of the experimental platform.



Name	Questionnaire	Last Attempt
Lastpass	Not started	<input type="radio"/>
Instagram	Not started	<input type="radio"/>
Microsoft	Not started	<input type="radio"/>
LinkedIn	Not started	<input type="radio"/>
Google	Done	<input checked="" type="radio"/>
SwissID	Done	<input checked="" type="radio"/>
Bitwarden	Not started	<input type="radio"/>
Reddit	Done	<input checked="" type="radio"/>
Twitter	Not started	<input type="radio"/>

Figure 5.2: Dashboard of the experimental platform.

dashboard page will be updated.

The architecture of the experimental platform consists of three main components, as illustrated in Fig. 5.1. The frontend is built using a Next.js JavaScript application that serves the content of the experiment platform and renders the application in the user’s browser. The backend is a Python FastAPI application that interacts with the database and runs the Playphish phishing framework. The frontend accesses the database through a RESTful API, while interaction with the phishing framework is done using the WebSocket protocol. The WebSocket protocol was chosen because the runtime phishing process requires a state to be kept, and Playphish must be able to push updates to the frontend. The application is hosted on a Docker container within the ETH infrastructure. A Nginx reverse proxy is used to forward incoming requests to our application and manage SSL encryption.

The following sections provide a comprehensive, implementation-level description of the various components of the experimental platform.

5.2 Frontend Javascript Application

The frontend application enables participants to interact with the experimental platform. It includes a landing page, consent page, main dashboard, questionnaire, and experiment view. Once participants have submitted their consent, a pseudonym in the form of a Universally Unique Identifier (UUID) is generated and assigned to the user. The user context is stored as a JSON Web Token (JWT) in a cookie. The dashboard, shown in Fig. 5.2, is the core of the platform, where participants can initiate new runtime phishing simulations and view an overview of their recent attempts.

For the development of the frontend application, we chose to use Next.js [21], a JavaScript framework based on React that allows for the creation of full-stack web applications. One of the main reasons for choosing Next.js was its built-in tools for routing, data fetching, and rendering. Additionally, the newest version of Next.js offers server-side rendering by default, which was particularly useful for communicating with the backend. It enabled access to the REST API from server components, eliminating the need to make the REST API public.

5.2.1 Authentication

Authentication is a crucial part of the application as it ensures the confidentiality of participant data on the experimental platform. To achieve this, a JWT token is created for each participant and encrypted before being stored in a secure cookie. This allows participants to return to the experiment as they remain authenticated until the cookie expires. We decided to use Auth.js [39] to handle authentication due to its secure and convenient integration with Next.js. The JWT token is used for all REST API requests that require user context and is securely passed in the HTTP Bearer header.

5.3 Python RESTful API

The RESTful API is built using a Python FastAPI [41] application and serves as a bridge between the frontend and the database. It also provides a route to retrieve information about the runtime phishing targets that must be displayed on the frontend. All routes, except for the route to create new participants, are protected. Access to these routes requires passing the JWT token as an HTTP Bearer header in the request. The JWT token contains the participant's pseudonym, which can be used to manipulate data in the database. It should be noted that the REST API does not need to be publicly accessible. All requests originate from the server-side of the JavaScript application, which minimizes the attack surface of the REST API and as most content can be rendered on the server, the application experiences a performance boost.

5.4 PostgreSQL Database

The experimental platform utilizes a PostgreSQL [15] database to store data related to the runtime phishing simulation, which can be analyzed later. The database is manipulated using the SQLAlchemy [2] ORM, and the schema is automatically generated with the database migration tool Alembic. The schema consists of four entities shown in Fig. 5.3

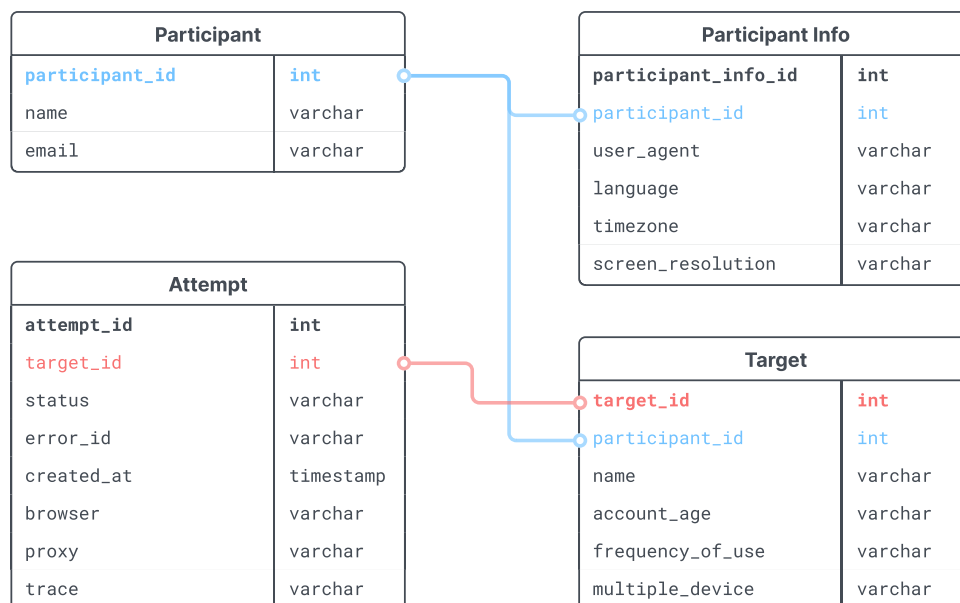


Figure 5.3: Database schema.

5.4.1 Participant

The participant entity contains all participants of the user study. In addition to the internal ID, the entity stores the name, which consists of the UUID used as a pseudonym for the participants, and an optional email address field.

5.4.2 Participant Info

As part of the user study described in Chapter 7, we recorded browser telemetry data from the participants. To ensure complete anonymization of participant data without losing information, we have decided to keep this information separate from the table containing their email addresses. As a result, the table containing the email addresses can be discarded after the study.

5.4.3 Target

A target entity is created for each participant and service provider, which contains the answers from the initial questionnaire (see Appendix A).

5.4.4 Attempt

The attempt entity stores the outcome of the runtime phishing simulation. The status field includes the values "success", "unsuccessful", or "error". If an error occurs during the simulation, an error ID is generated and persisted for later debugging. In addition to the outcome of the attempt, the browser and proxy used by the Playphish phishing framework, as well as the trace of the authentication attempt, are stored.

5.5 Python WebSocket API

The WebSocket API serves as an interface between the frontend and the Playphish phishing framework. During a runtime phishing scenario, multiple rounds of communication occur between the phishing framework and the phishing site. Fig. 5.4 provides an example of this. The legitimate website may initially request the user's username and password and, upon verification, prompt the user to enter a second factor. Communication should be stateful and bidirectional. The WebSocket protocol is well-suited for this purpose. Fig. 5.4 and Fig. 5.5 demonstrate the practical application of the WebSocket protocol in dynamically altering content on the experimental platform.

5.6 Deployment

The experimental platform enables the creation of containers for the different components using Docker Compose. These containers are then deployed to a virtual machine within the ETH infrastructure. A Nginx [43] reverse proxy is used on the VM to forward all requests to the corresponding containers. This approach allows for the frontend and WebSocket API to be served on the same domain. Additionally, the reverse proxy is used to handle SSL encryption and decryption.

5.7 Security Considerations

The deployed security measures are primarily designed to address an adversary model where the attacker does not have access to the machine where the experimental platform runs. It is crucial to note that if an attacker gains access to the machine, they could potentially extract data and credentials, bypassing some of these security measures. However, our focus has been on mitigating threats from external attackers.

Several security considerations were taken into account during the development of the experimental platform. This section highlights the two main points:

- Securing data persisted in the database
- Securing credentials during the phishing simulation

To maintain data confidentiality, the REST API has been made private. Access to the REST endpoint is restricted to within the docker network. Additionally, all routes that involve creating, reading, or updating user data require a JWT token for protection. To ensure the security of participants' credentials during the runtime phishing simulation, we employed the WSS (WebSockets over SSL/TLS) protocol. As a result, all messages transmitted between the client and the backend via the WebSocket protocol are encrypted.

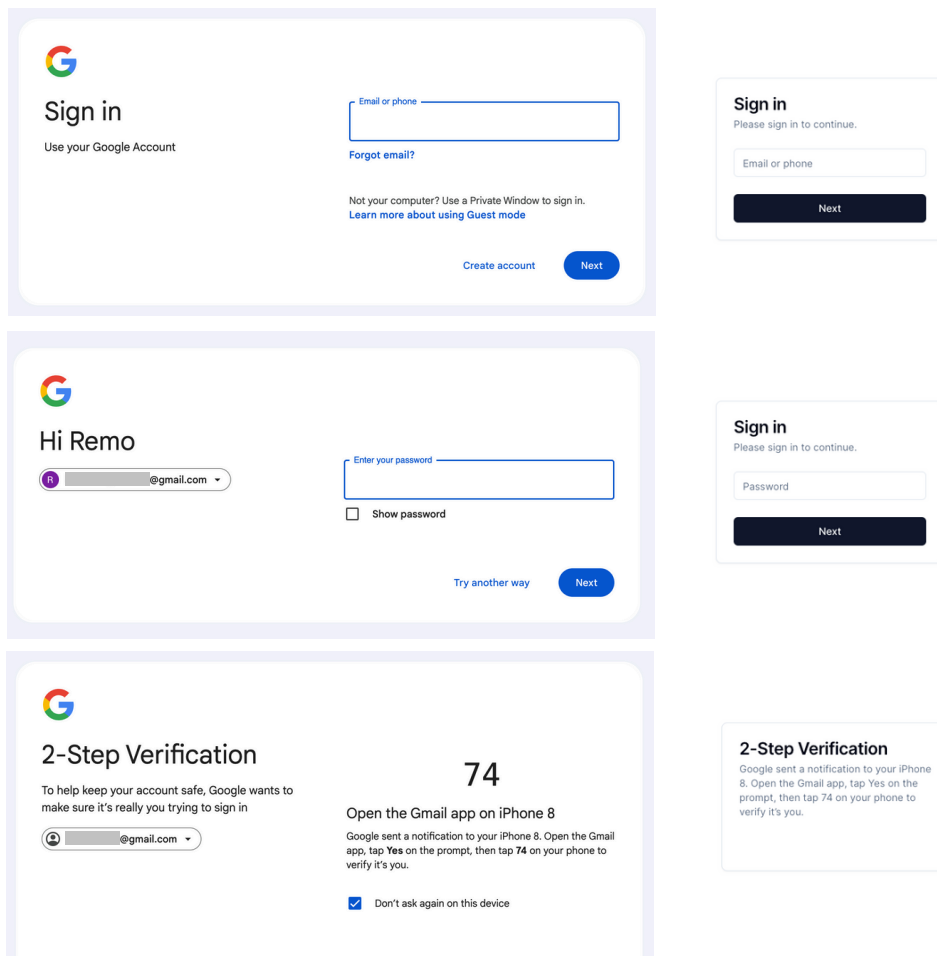


Figure 5.4: The left-hand screenshots depict the steps of a Google login, while the right-hand side displays the corresponding steps shown to participants on the experimental platform. The underlying messages sent through the WebSocket are shown in Fig. 5.5.



Figure 5.5: Messages sent through the WebSocket protocol between the frontend and Playphish. The corresponding screenshots are shown in Fig. 5.4.

Chapter 6

Data Collection

This chapter focuses on the data collection stage conducted as part of our thesis. The first part gives a high-level overview, followed by a more detailed explanation including the results.

6.1 Overview

In the data collection stage, new accounts were created on eleven high-traffic online services using an artificial identity. Two-factor authentication (2FA) was enabled for each account, either through OTPs or SMS verification codes. These accounts were then used to simulate runtime phishing attacks against the eleven services using the Playphish phishing framework. The attacks were simulated with increasing levels of sophistication until they succeeded. This provided valuable insights into the effectiveness of the deployed protection techniques and the level of care taken by service providers against such attacks.

6.2 Experimental Setup

The following sections provide an overview of the settings employed for the collection of data. Playphish was run twice for each service and browser, once in headful mode and once in headless mode (i.e. the controlled browser was run with and without a graphical user interface). All runs were carried out on an Apple MacBook from the ETH network. The phishing framework was initially run with default settings, including no user behavior. If the attempt was unsuccessful, the attacker's level of sophistication was increased by incorporating various modifications to Playphish. Table 6.2 provides a complete list of the base settings and modifications used.

6.2.1 Targets

We aimed to have a diverse selection of targets from a wide range of industries. Based on the findings of the Anti-Phishing Working Group (APWG) in their latest phishing trends report [14], we selected Google and Microsoft as representatives of the SAAS/webmail industry and the platforms LinkedIn, Instagram, Reddit, and Twitter as representatives of the social media industry. We also wanted to test service providers from industries with higher security standards. For this reason, we decided to include the two password managers Lastpass and Bitwarden, and the crypto exchange platforms Coinbase and Binance. Table 6.1 provides a comprehensive overview of the targets included in the data collection and evaluation stage, as detailed in Chapter 7. It lists the two-factor authentication (2FA) methods supported by each target.

6.2.2 Browser

Playwright supports a range of browsers, including Chromium, Firefox, and Safari. To ascertain the impact of the browser choice on the success of a runtime phishing attack, a series of simulations were

conducted utilizing all three browsers. By default, Playwright is configured to run in headless mode, which means that it does not have a graphical user interface. In the context of a runtime phishing attack, running the browser in headless mode is preferred as it is easier to scale and more resource-efficient, as no graphical resources are required. However, it is generally easier to detect headless browsers. Therefore, we also aimed to investigate the difference between running the attack in headful or headless mode.

6.2.3 Test Accounts

As part of the initial data collection stage, we created test accounts for the eleven services using an artificial identity. All services, except SwissID, allowed the activation of OTP 2FA. To generate verification codes, we used an authenticator app. For SwissID, we obtained a prepaid SIM card to enable the use of SMS verification codes as a second factor. The accounts were created from the ETH network on an Apple Macbook using the Google Chrome browser.

	Category	Supported 2FA Methods
Twitter	Social Media	SMS, OTP, Security Key
Instagram	Social Media	SMS, OTP
Bitwarden	Password Manager	Email, OTP, Security Key
Lastpass	Password Manager	Push, OTP, Security Key (Premium only)
Reddit	Social Media	OTP
LinkedIn	Social Media	SMS, OTP
Google	SAAS / Webmail	Push, SMS, OTP, Security Key
SwissID	Digital Identity	Push, SMS, Cross-off List ¹
Microsoft	SAAS / Webmail	Push, SMS, Email, OTP, Security Key
Coinbase	Crypto Exchange	Push, SMS, Email, OTP, Security Key
Binance	Crypto Exchange	SMS, Email, OTP, Security Key

Table 6.1: Overview of the targets used in the data collection and evaluation.

Modification	Settings
Base	No delay between actions
	Default user agent
	No mouse movement
	No keyboard typing
User behavior	Random mouse movement between actions
	Delay between actions
	Inputs are filled sequentially (Simulate typing)
	Human-like clicking (Delay between mouse down and mouse up event)
Hide automation	Use authentic user agents
	Set <code>navigator.webdriver</code> to false
Playwright stealth	Use Playwright stealth plugin
Browser fingerprint	Spoof HTTP headers

Table 6.2: Overview of settings used to run Playphish during the assessment phase.

¹A Cross-off list is a printable list of codes that can be used instead of SMS-based codes. Each code can only be used once.

6.3 Results

Table 6.3 shows the results of the data collection stage with base settings. Six out of the eleven services were successfully logged into without any modifications for all three browsers. When using the Chromium browser, Reddit and Microsoft blocked the login attempts, while Instagram displayed a warning (but did not block the attempt) stating that an automation tool had been detected. This was because Chromium in Playwright is using "HeadlessChrome" in its user agent string. The issues were resolved by replacing it with just "Chrome".

	Twitter	Instagram	Bitwarden	Lastpass	Reddit	LinkedIn	Google	SwissID	Microsoft	Coinbase	Binance
Chromium	●	●	●	●	●	●	-	●	●	-	-
(headless)	●	●	●	●	-	●	-	●	-	-	-
Firefox	●	●	●	●	●	●	●	●	●	-	●
(headless)	●	●	●	●	●	●	●	●	●	-	●
Safari	●	●	●	●	●	●	-	●	●	-	●
(headless)	●	●	●	●	●	●	-	●	●	-	●

Table 6.3: Comparison of a runtime phishing attack against eleven high-traffic online services with default settings from different browsers. Symbols: ● the attack was successful, ● a CAPTCHA prevented the attack sometimes, and - the attack was blocked by the service provider.

6.3.1 Google

Google successfully blocked the authentication attempts when using Chromium or Safari. The issue for Chromium was solved by applying the "hide automation" modification using the command-line option `--disable-blink-features=AutomationControlled`. No other modifications had any effect on the result. Using "Playwright stealth" together with "Hide automation" made the issue worse and it stopped working. None of the modifications worked for Safari.

6.3.2 Coinbase

All runtime phishing attempts for Coinbase were rejected. Coinbase requires users to verify new devices. This is done by opening a link received via email on the new device. The system verifies that the device attempting to authenticate matches the device that opened the link.

6.3.3 Binance

Binance rejected all authentication attempts originating from Chromium by displaying a CAPTCHA. For Firefox and Safari, Binance accepted some attempts and rejected others with a CAPTCHA.

Chapter 7

Evaluation

This chapter focuses on the user study conducted as part of our thesis. The first part gives a high-level overview of the study, followed by a more detailed explanation including the results of the user study.

7.1 Overview

We developed an experimental platform that enabled us to simulate runtime phishing attacks with real user accounts. To properly validate the protection techniques used by service providers, we assessed our phishing framework across multiple accounts with varying demographics, such as account age or "internal risk score". We simulated runtime phishing attacks from different networks and browsers – all factors that may impact the service provider's decision to accept or reject our authentication attempt.

Participants were recruited through word-of-mouth from our environment, provided they had an account for at least one of the eleven services. They were given the option to participate in the study for each service individually. The evaluation started with a short questionnaire about the account age, frequency of use, and the preferred device for accessing the account. Participants were instructed to log in to their accounts using the login screen on the experimental platform. Subsequently, a runtime phishing attack was carried out in the background, with the simulation options (i.e. browser and network) chosen randomly. The results of the phishing attempt, including success or failure, were logged into a database.

7.2 Scope and Attacker Model

It is important to note that this thesis focuses solely on evaluating the technical capabilities of service providers in preventing phishing attacks. The social engineering aspect, which involves tricking users into visiting a phishing site, is not taken into consideration. The experiments conducted assume that the user has followed a malicious link, unaware of the fraud, willingly to provide their credentials to the phishing site. Consequently, the user should comply with all prompts and requests encountered during the authentication process. Therefore, we did not visually adapt the login screen presented to the participants on the experimental platform to match the login screen of the legitimate service.

It is assumed that the attacker cannot compromise the user's computer or smartphone, such as installing malicious software in the user's browser to directly access session cookies. Additionally, the attacker would not be able to generate 2FA verification codes on their own.

7.3 Study design

Participants were initially presented with an overview of the subject of runtime phishing, including the objective and methodology of the study. Detailed information was available on the study website,

which participants were encouraged to review before giving their consent. The experiment proceeded as follows:

- (1) Upon visiting the study website, participants were shown the consent form and were asked to express their consent by marking a checkbox.
- (2) Consenting participants were then shown the list of service providers. By selecting one of these, the experiment started.
- (3) Participants then answer optional questions about their account (See Appendix A).
- (4) After the questionnaire, the participants are presented a login form for the selected provider, simulating the runtime phishing attack.
 - (a) In the background, an instance of the Playphish phishing framework was started, forwarding the credentials immediately to the authentic service.
- (5) If the authentication succeeded or the runtime phishing attempt was prevented from the service provider, participants return to the list of providers.

The following data was recorded throughout the experiment: for participants who had consented to participate, browser telemetry data was stored from the browser they used to visit the study website. This included the user agent, language, time zone, and screen resolution. The answers to the questionnaire (see Appendix A) were stored for each account. Upon completion of the phishing attempt, a boolean flag was recorded to indicate success. Furthermore, the browser and network used by the phishing framework during the simulation are recorded, along with a trace of the authentication. This trace consists of a list of the steps encountered during the login process, which provides insights into the two-factor authentication (2FA) methods and any additional verification requested from the user.

7.3.1 Participants

Participants were recruited through word-of-mouth from our environment. To ensure transparency and a thorough understanding of the study, we provided detailed information to potential participants. The same information was also available in detail on the study website. We recruited 15 participants from various regions of Switzerland having an account for at least one of the eleven target services.

7.3.2 Browser

The Playphish framework was run in headless mode for all browsers. Based on the findings of the data collection stage, we selected Firefox and Chromium for the simulation. For each attempt, we randomly selected one of the supported browsers. No alterations were made to the default settings of Firefox. Furthermore, no user behavior was replicated. Google and Microsoft were required to utilize Firefox exclusively in the experiment due to technical difficulties encountered with Chromium during the user study.

Chromium

The following modifications were made to the default settings of Chromium. This was necessary to conceal the automation-controlled browser.

- Use option `--disable-blink-features=AutomationControlled`
- Replaced the term "HeadlessChrome" with just "Chrome" in the user agent string

7.3.3 Proxy

For the evaluation, three different proxies were used. The purpose of this was to measure the network's influence on whether the login from the simulated attack is allowed or rejected by the service providers. The chosen proxies were:

- **default:** Some participants have already accessed their accounts from this network
- **residential:** The participants have not previously accessed their accounts from this network
- **germany:** The participants have not previously accessed their accounts from this network *and* the network is located in a different country

Previous research has demonstrated that high-traffic online services (e.g. Amazon, Google, LinkedIn, and Facebook) use Risk-based Authentication (RBA) and consider the IP address of the authentication attempt to determine if additional verification is necessary [56]. It has been shown that, for services that use RBA, accessing them from a different country will always result in an additional verification request. Therefore, we added a proxy to the phishing framework that was hosted on a DigitalOcean VPS in Frankfurt, Germany. Additionally, a residential proxy was included to test if services are more likely to accept an authentication request from a residential proxy rather than a data center proxy.

	Geolocation	ISP	Description
default	Zurich, Switzerland	ETH Zürich	workplace proxy, same country
residential	Zürich, Switzerland	Init7	residential proxy, same country
germany	Frankfurt, Germany	DigitalOcean	data center proxy, different country

Table 7.1: Overview of proxies used for the phishing simulation experiment.

7.4 Results

In the initial phase of the evaluation, we conducted a preliminary assessment using our accounts. As in the data collection phase, we were unable to complete an attack against Coinbase. Additionally, all attempts to access Binance were prevented due to the presence of a CAPTCHA. Consequently, we decided to exclude both services from the evaluation with other participants.

Fig. 7.1 presents the findings of the user study for the different services. Of the remaining nine services, six of them were susceptible to all runtime phishing attempts. For three services, a CAPTCHA was triggered, mainly when running the attack from the German data center proxy (see Fig. 7.2). However, CAPTCHAs were the only technique employed by service providers to counteract runtime phishing attacks. There was no apparent correlation between the chosen browser and the success rate of the phishing simulation. The following sections will provide a more detailed examination of the resistance observed during the user study.

7.4.1 CAPTCHAs

As previously stated, the use of CAPTCHAs was the primary factor in the rejection of phishing attempts. Except for one instance, all attempts originated from the German data center proxy, as illustrated in Fig. 7.2. Google was the only service where a CAPTCHA was triggered from the residential proxy. For Google, two participants encountered a CAPTCHA. Both participants, for whom the attempt failed, stated in the questionnaire that they use their Google account less than monthly.

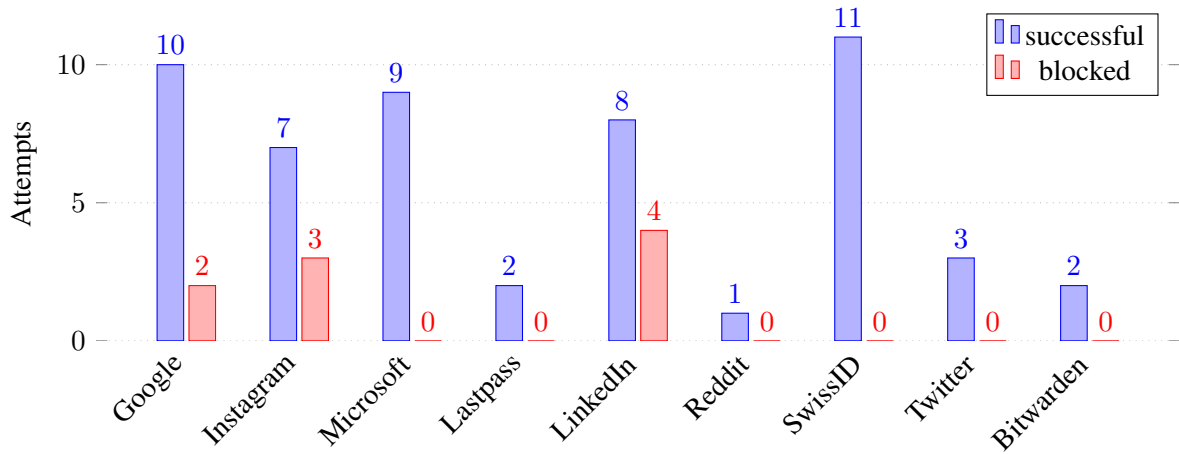


Figure 7.1: Number of unique authentication requests per service that were accepted or blocked.

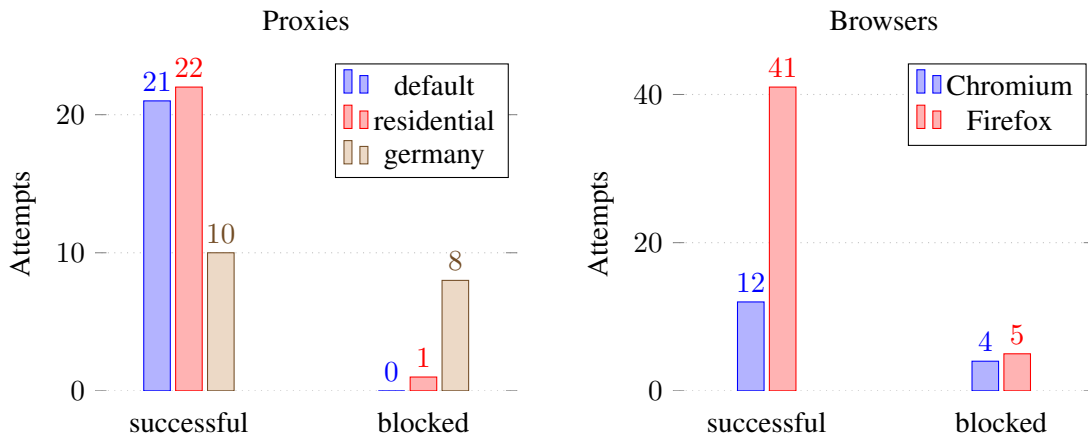


Figure 7.2: Comparison of successful and blocked authentication attempts for different browser and network settings.

7.4.2 Notifications

Notifications have been one of the most prominent elements encountered during the study, with the majority of the services offering some form of alert following a phishing attempt. However, these notifications were sent to the participants after the login was successful, which means that they only protect users from phishing to a certain extent. A selection of these notifications can be seen in Fig. 7.3. All login attempts from which the notifications originated were made via the experimental platform. This implies that they were carried out on a Linux machine with either Firefox or Chromium. As can be seen, it was possible to fake a false operating system and browser for all services except Google. From all nine services included in the user study, except for Google and Microsoft, it was observed that all accepted the spoofed platform and browser and utilized them within the notifications sent to the participants.

7.4.3 Other Resistance

Microsoft offers a passwordless authentication system, which utilizes the Microsoft Authenticator app. Following the submission of a username, users receive a notification from the authenticator app, prompting them to either enter a PIN code or verify their identity through biometric information. However, it has been demonstrated that all runtime phishing attacks against participants with the passwordless option activated were successful.

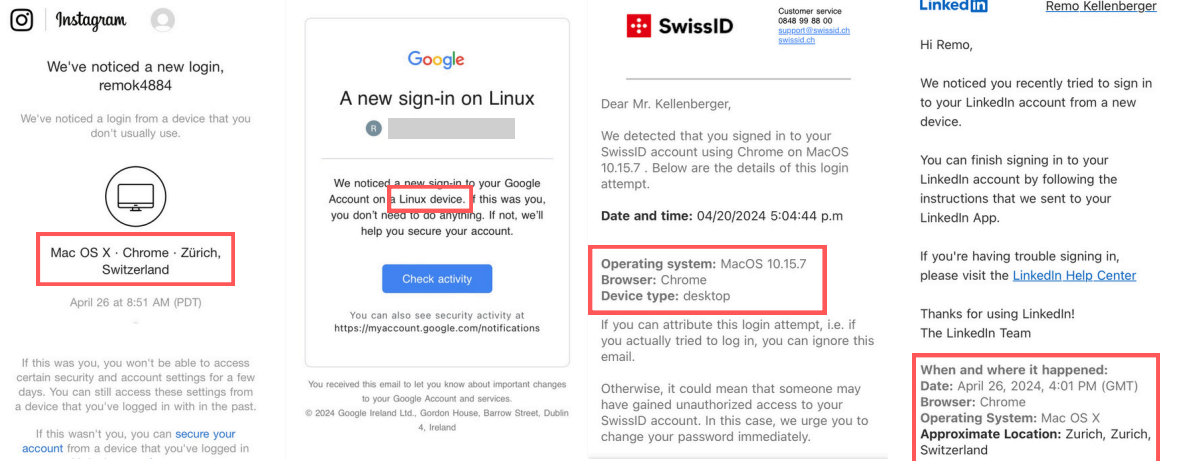


Figure 7.3: Notifications sent to participants following the successful completion of a runtime phishing attempt.

Chapter 8

Discussion

This chapter will first discuss the results in the context of the research questions posed in Chapter 3. In addition, this chapter will briefly address the subject of ethics

Findings related to Q1. The objective of research question Q1 was to assess the extent to which service providers are concerned about runtime phishing. To this end, we developed a custom runtime phishing framework, called Playphish, which was used to simulate such attacks against eleven high-traffic service providers. In the initial data collection phase, we ran the framework against newly created test accounts on the eleven services. The Playphish framework was employed with the default settings, which did not include any human behavior or attempts to disguise the fact that the login requests originated from an automated browser. Consequently, the tool could have been easily identified and blocked. For instance, a check of the user agent string would have sufficed to identify the requests with Chromium in headless mode. Reddit has employed a similar approach. In addition, the `navigator.webdriver` property could have been checked, which is set by default for headless Chromium. Nevertheless, the runtime phishing attack was successful for six out of eleven targets. Furthermore, a switch to Firefox resulted in the attack being successful against nine out of eleven targets, even in headless mode.

The evaluation conducted on real user accounts revealed comparable observations. All unsuccessful attempts were attributed to the use of CAPTCHAs. However, CAPTCHAs offer only limited protection in the context of runtime phishing. Some weak variants can be solved automatically, while for those that cannot be solved automatically, there are so-called CAPTCHA farms where real people solve the CAPTCHAs for a small fee. Nevertheless, in the context of runtime phishing, this is not a necessity, as more sophisticated phishing toolkits could simply forward the CAPTCHA to the victim for resolution.

Findings related to Q2. The objective of research question Q2 was to analyze the techniques employed by services to protect their users from runtime phishing. Contrary to our expectations, we did not find that user behavior influenced the success rate of a runtime phishing attack. The origin of the attack was much more decisive. Of all the attacks that originated from the ETH network or a residential network, only one was unsuccessful, despite most participants of the user study were never connected to these networks.

Nevertheless, we have observed a technique in one service that successfully defended against all attacks using our phishing framework. Coinbase users must confirm each new device via a link they receive by email. The distinctive feature of this link is that it is only valid if it is opened on the same device that is used for authentication. Consequently, an attacker positioned between the victim and Coinbase is unable to log in on behalf of the victim. The most straightforward method for achieving this would be for Coinbase to set a cookie containing a unique value when a new device attempts to authenticate. Upon opening the link, Coinbase could then verify that the cookie matches. However, Coinbase has taken a different approach. Our tests have demonstrated that Coinbase employs some form of advanced device fingerprinting, enabling the verification of a new device even when the link is opened in a different browser on the same device.

Coinbase has demonstrated the efficacy of a method to protect against runtime phishing attacks. However, this approach is constrained by its usability. Users are required to have access to the link on the device they wish to sign in. This necessitates either logging in to the email provider or manually entering the link.

Another technique that has been identified during the research is that some services require step-up verification for sensitive account actions. This implies that an attacker would gain access to the account, but as soon as they attempted to perform a sensitive action, they would be prompted for additional verification.

Findings related to Q3. Research question Q3 deals with the costs associated with developing a successful runtime phishing attack against a service provider. Tools such as Evilginx or Modlishka facilitate the process of setting up a runtime phishing attack. There is no need to concern oneself with the construction of a replica of a site. Evilginx is even able to automatically install a TLS certificate. The targets are configured using so-called phishlets. Through targeted research, it is possible to find ready-made phishlets for the majority of major service providers online.

The development of Playphish has demonstrated that the costs associated with the creation of a custom phishing toolkit are also relatively low. No sophisticated user behavior was required, nor were any significant alterations made to the default configurations of the browser automation tool.

One reason why online services do not attempt to address the issue of runtime phishing may be the complexity of the problem. The service providers may be allocating their resources primarily towards the adoption of FIDO. FIDO offers an alternative to password-based authentication that is considered to be phishing-resistant. However, it will be a time before this is established. Of the 11 services that were analyzed, seven already support FIDO passkeys. However, all seven of these services permit the use of a weaker alternative, such as backup keys or other two-factor authentication methods. Google's Advanced Protection Program [34] is the only service that restricts its users to the sole use of security keys. Furthermore, it is important to note that FIDO's latest set of specifications, which enables users to utilize their smartphones as authenticators, is still in draft form.

8.1 Ethics and Security

One of the primary objectives of this research, and in particular the user study, was to adhere to the ethical guidelines that have been established within this field of study. The participants in the study were carefully selected and provided with comprehensive information about the study. To ensure transparency and a thorough understanding of the study, detailed information is provided through personal briefings and the experimental platform.

8.1.1 Data Protection

It should be noted that, except for browser telemetry data, no personal data was collected throughout the experiment. It is important to emphasize that no login information was gathered from participants, and that credentials were solely used to simulate a runtime phishing attack. Participants were assigned a random identifier to ensure pseudonymisation of the data collected during the study. The experimental platform was carefully developed and thoroughly tested to ensure the security of data at rest and in transit.

8.1.2 Risks and Countermeasure

The primary concern for participants was the highly unlikely scenario of their accounts being suspended due to automated behavior detection or as a security precaution. To address and minimize this risk,

the following strategies have been employed. The Playphish phishing framework was designed to mitigate the risk of being exposed as an automation-controlled browser. During the data collection stage, the framework was sufficiently tested, and the experimental platform was tested with our authentic accounts. Participants had the option to withdraw from the study at any time. Each participant was able to decide for each specific service provider whether to participate or not according to their comfort and risk tolerance for each account. Furthermore, the user study was constantly monitored, and if a participant encountered account suspension, the service provider would have been removed immediately from the study.

In evaluating the potential risks, it was determined that the anticipated social and scientific benefits greatly outweighed the potential risks. The study not only enhanced scientific comprehension within the field but also has the potential to significantly enhance the security of online services, ultimately benefiting users by strengthening their protection.

Chapter 9

Conclusion

In this thesis, we aimed to analyze the resilience against runtime phishing attacks. Runtime phishing attacks represent a novel form of phishing attacks that can bypass two-factor authentication (2FA). All 2FA schemes except FIDO are susceptible to a common vulnerability. This vulnerability is the inability to verify whether the user is interacting with a legitimate site or a phishing site. With FIDO, there exists a phishing-proof alternative to password-based authentication. However, FIDO's adoption is far from complete. One limitation of services that currently support FIDO for authentication is that they are susceptible to downgrade attacks. Consequently, runtime phishing remains a significant threat.

To study the effectiveness of runtime phishing attacks in the wild, we have developed a runtime phishing framework, called Playphish. Playphish employs the browser automation tool Playwright to automatically forward credentials, including 2FA tokens, between the victim and the legitimate site in real time.

In the initial data collection stage, Playphish was tested against eleven high-traffic service providers, including Google, Microsoft, Instagram, and LinkedIn. The tests demonstrated that Playphish was able to successfully run attacks against nine of the eleven services using only the default configuration of Playwright and without implementing user behavior. All attempts to phish Coinbase were blocked due to a mechanism built into the Coinbase authentication system. To verify a new device, Coinbase requires the user to click on a link sent via email on the same device.

In the second stage, we sought to validate our findings from the data collection stage using real user accounts. To this end, we implemented an experimental platform where participants in our user study were able to simulate runtime phishing attacks. The simulations were conducted from different networks and with different browsers, which may influence the success of a runtime phishing attack. The only resistance encountered was bot detection. For three services, some attempts (mainly those originating from a German data center network) were blocked through a CAPTCHA.

This research has demonstrated that runtime phishing attacks remain still highly effective against popular services. It has also shown that implementing such an attack is relatively cost-effective. Although there are few effective methods for protecting against such attacks, the example of Coinbase has demonstrated a method that is effective in practice. Moreover, the FIDO standard offers the potential for a phishing-resistant authentication method. However, there are still some limitations and it is not widely known among non-experts.

Appendix A

Account questionnaire

During the user study, participants were asked to answer the following questions about their account:

Whats is the approximate age of your account?

- (a) Less than a week
- (b) Between 1 week and 3 months
- (c) Between 3 months and 1 year
- (d) Older
- (e) No answer

How often do you use your account?

- (a) Daily
- (b) Weekly
- (c) Monthly
- (d) Less than monthly
- (e) No answer

Are you using your account on multiple devices (e.g., Laptop, Smartphone)?

- (a) Yes
- (b) No
- (c) No answer

Bibliography

- [1] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. Visualphishnet: Zero-day phishing website detection by visual similarity. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1681–1698, 2020.
- [2] Michael Bayer. Ssqlalchemy. In Amy Brown and Greg Wilson, editors, *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012.
- [3] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, 2012.
- [4] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567, 2012.
- [5] Internet Crime Compliant Center. Internet crime report 2023. https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf, 2023. Accessed: April 2024.
- [6] Pan Chan and Trevor Haskell. Reelphish. <https://github.com/mandiant/ReelPhish>. Accessed: April 2024.
- [7] Software Freedom Conservancy. Selenium. <https://www.selenium.dev/>. Accessed: April 2024.
- [8] Microsoft Corporation. Playwright. <https://playwright.dev/python/>. Accessed: April 2024.
- [9] Nguyen Ngoc Diep, Sungyoung Lee, Young-Koo Lee, and HeeJo Lee. Contextual risk-based access control. *Security and Management*, 2007:406–412, 2007.
- [10] Piotr Duszyński. Modlishka. <https://github.com/drklwi/Modlishka>. Accessed: April 2024.
- [11] Mike Felch. Credsniper. <https://github.com/ustayready/CredSniper>. Accessed: April 2024.
- [12] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is fido2 the kingslayer of user authentication? a comparative usability study of fido2 passwordless authentication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 268–285, 2020.
- [13] Kuba Gretzky. Eviglinx. <https://github.com/kgretzky/evilginx2>. Accessed: April 2024.
- [14] Anti-Phishing Working Group. Phishing activity trends report q4 2023. <https://apwg.org/trendsreports/>, 2024. Accessed: April 2024.

- [15] The PostgreSQL Global Development Group. PostgreSQL. <https://www.postgresql.org/>. Accessed: April 2024.
- [16] FIDO Alliance Inc. Fido alliance. <https://fidoalliance.org/>. Accessed: April 2024.
- [17] FIDO Alliance Inc. Client to authenticator protocol (ctap). <https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html>, 2022. Accessed: April 2024.
- [18] FIDO Alliance Inc. Passkeys. <https://fidoalliance.org/passkeys/>, 2022. Accessed: April 2024.
- [19] FIDO Alliance Inc. White paper: Multi-device fido credentials. <https://fidoalliance.org/white-paper-multi-device-fido-credentials/>, 2022. Accessed: April 2024.
- [20] Keeper Security Inc. Password management report: Unifying perception with reality. <https://www.keepersecurity.com/password-management-report-unifying-perception-with-reality/>, 2023. Accessed: April 2024.
- [21] Vercel Inc. Next.js - the react framework for the web. <https://nextjs.org/>. Accessed: April 2024.
- [22] World Wide Web Consortium Inc. Web authentication: An api for accessing public key credentials. <https://www.w3.org/TR/webauthn/>, 2021. Accessed: April 2024.
- [23] Ankit Kumar Jain, Brij B Gupta, et al. Phishing detection: analysis of visual similarity based approaches. *Security and Communication Networks*, 2017, 2017.
- [24] Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. {Sound-Proof}: Usable {Two-Factor} authentication based on ambient sound. In *24th USENIX security symposium (USENIX security 15)*, pages 483–498, 2015.
- [25] Mahmoud Khonji, Youssef Iraqi, and Andrew Jones. Phishing detection: a literature survey. *IEEE Communications Surveys & Tutorials*, 15(4):2091–2121, 2013.
- [26] Brian Kondracki, Babak Amin Azad, Oleksii Starov, and Nick Nikiforakis. Catching transparent phish: Analyzing and detecting mitm phishing toolkits. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 36–50, 2021.
- [27] KrebsonSecurity. Google: Security keys neutralized employee phishing. <https://krebsonsecurity.com/2018/07/google-security-keys-neutralized-employee-phishing/>, 2018. Accessed: April 2024.
- [28] Ponnurangam Kumaraguru, Yong Rhee, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. Protecting people from phishing: the design and evaluation of an embedded training email system. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 905–914, 2007.
- [29] Ponnurangam Kumaraguru, Yong Rhee, Steve Sheng, Sharique Hasan, Alessandro Acquisti, Lorrie Faith Cranor, and Jason Hong. Getting users to pay attention to anti-phishing education: evaluation of retention and transfer. In *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 70–81, 2007.

-
- [30] Daniele Lain, Kari Kostiaainen, and Srdjan Čapkun. Phishing in organizations: Findings from a large-scale and long-term study. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 842–859. IEEE, 2022.
- [31] Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*, 2018.
- [32] Google LLC. Google safe browsing. <https://safebrowsing.google.com/>. Accessed: April 2024.
- [33] Google LLC. Puppeteer. <https://pptr.dev/>. Accessed: April 2024.
- [34] Google LLC. Advanced protection program. <https://landing.google.com/advancedprotection/>, 2018. Accessed: April 2024.
- [35] Google LLC. Better protection against man in the middle phishing attacks. <https://security.googleblog.com/2019/04/better-protection-against-man-in-middle.html>, 2019. Accessed: April 2024.
- [36] Google LLC. Google authenticator app. <https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2>, 2024. Accessed: April 2024.
- [37] Jian Mao, Wenqian Tian, Pei Li, Tao Wei, and Zhenkai Liang. Phishing-alarm: Robust and efficient phishing detection via page component similarity. *IEEE Access*, 5:17020–17030, 2017.
- [38] Tommy Mysk and Talal Haj Bakry. Can a tesla stop phishing and social engineering attacks? <https://www.mysk.blog/2024/03/10/tesla-phone-key/>, 2024. Accessed: April 2024.
- [39] Balázs Orbán, Thang Vu, Nico Domino, and Lluís Agusti. Auth.js - authentication for the web. <https://authjs.dev/>. Accessed: April 2024.
- [40] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 417–434. IEEE, 2019.
- [41] Sebastián Ramírez. Fastapi. <https://fastapi.tiangolo.com/>. Accessed: April 2024.
- [42] Ken Reese, Trevor Smith, Jonathan Dutson, Jonathan Armknecht, Jacob Cameron, and Kent Seamons. A usability study of five {two-factor} authentication methods. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 357–370, 2019.
- [43] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux J.*, 2008(173), sep 2008.
- [44] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*, 2017.
- [45] Duo Security. One-tap authentication with duo push. <https://duo.com/product/multi-factor-authentication-mfa/authentication-methods/duo-push>. Accessed: April 2024.
- [46] Steve Sheng, Bryant Magnien, Ponnurangam Kumaraguru, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. Anti-phishing phil: the design and evaluation of a game that teaches people not to fall for phish. In *Proceedings of the 3rd symposium on Usable privacy and security*, pages 88–99, 2007.

- [47] Simon Stockhardt, Benjamin Reinheimer, Melanie Volkamer, Peter Mayer, Alexandra Kunz, Philipp Rack, and Daniel Lehmann. Teaching phishing-security: which way is best? In *ICT Systems Security and Privacy Protection: 31st IFIP TC 11 International Conference, SEC 2016, Ghent, Belgium, May 30-June 1, 2016, Proceedings 31*, pages 135–149. Springer, 2016.
- [48] Yuanyi Sun, Sencun Zhu, Yan Zhao, and Pengfei Sun. A user-friendly two-factor authentication method against real-time phishing attacks. In *2022 IEEE Conference on Communications and Network Security (CNS)*, pages 91–99, 2022.
- [49] Muraena Team. Muraena. <https://github.com/muraenateam/muraena>. Accessed: April 2024.
- [50] Issa Traore, Isaac Woungang, Mohammad S. Obaidat, Youssef Nakkabi, and Iris Lai. Combining mouse and keystroke dynamics biometrics for risk-based authentication in web environments. In *2012 Fourth International Conference on Digital Home*, pages 138–145, 2012.
- [51] Enis Ulqinaku, Hala Assal, AbdelRahman Abdou, Sonia Chiasson, and Srdjan Capkun. Is real-time phishing eliminated with FIDO? social engineering downgrade attacks against FIDO protocols. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3811–3828. USENIX Association, August 2021.
- [52] Enis Ulqinaku, Daniele Lain, and Srdjan Capkun. 2fa-pp: 2nd factor phishing prevention. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 60–70, 2019.
- [53] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 294–311. Springer, 2003.
- [54] Ding Wang, Haibo Cheng, Ping Wang, Xinyi Huang, and Gaopeng Jian. Zipf’s law in passwords. *IEEE Transactions on Information Forensics and Security*, 12(11):2776–2791, 2017.
- [55] Rick Wash, Emilee Rader, Ruthie Berman, and Zac Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 175–188, Denver, CO, June 2016. USENIX Association.
- [56] Stephan Wiefeling, Luigi Lo Iacono, and Markus Dürmuth. Is This Really You? An Empirical Study on Risk-Based Authentication Applied in the Wild. In *34th IFIP TC-11 International Conference on Information Security and Privacy Protection (IFIP SEC 2019)*, volume 562 of *IFIP Advances in Information and Communication Technology*, pages 134–148. Springer International Publishing, June 2019.
- [57] Jiliang Zhang, Xiao Tan, Xiangqi Wang, Aibin Yan, and Zheng Qin. T2fa: Transparent two-factor authentication. *IEEE Access*, 6:32677–32686, 2018.
- [58] Yue Zhang, Jason I Hong, and Lorrie F Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on World Wide Web*, pages 639–648, 2007.