Doubly Dangerous: Evading Phishing Reporting Systems by Leveraging Email Tracking Techniques

Anish Chand, *Louisiana State University;* Nick Nikiforakis, *Stony Brook University;*
Phani Vadrevu, *Louisiana State University*

https://www.usenix.org/conference/usenixsecurity25/presentation/chand

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Doubly Dangerous: Evading Phishing Reporting Systems by Leveraging Email Tracking Techniques

Anish Chand
Louisiana State University

Nick Nikiforakis
Stony Brook University

Phani Vadrevu
Louisiana State University

Abstract

Given the significant threat posed by email as a highly prevalent phishing attack vector, we undertake the first study focused on real-world phishing email reporting systems. Our key idea in performing this study is to repurpose email tracking, a well-known privacy threat vector, for profiling and evading anti-phishing systems employed by popular email services. Our results show that the reporting systems of all major email services we tested are vulnerable to evasive phishing attacks affecting more than 2 billion users worldwide. We propose several countermeasures that email service operators can adopt to help ameliorate this issue in the future. We disclosed our findings to the affected email providers which resulted in remedial changes and a vulnerability reward.

1 Introduction

Phishing remains a persistent challenge in cybersecurity. Efforts to address phishing have largely focused on specific sub-problems, such as developing advanced machine learning-based phishing detectors [24, 34, 36, 37, 38, 39] and analyzing phishing attacks in the wild [27, 33, 45, 51, 56]. However, real-world phishing reporting systems have received relatively little attention. Specifically, there has been limited research on how the deployed phishing detectors used by large service providers respond to user-reported phishing websites. This is a critical gap since any discernible patterns in the behavior of these anti-phishing systems in response to user reports can be discovered and exploited by attackers. For instance, if a phishing detector's behavior is identifiable, attackers could leverage cloaking techniques to display harmless content to it while maintaining malicious behavior for regular users [57].

The limited prior research in the domain of phishing reporting has already demonstrated the ability to create evasive phishing websites that bypass security crawlers designed to operate incognito [25, 26, 28, 43, 44]. However, a significant limitation of this prior work is its exclusive focus on web-based phishing reporting systems, which are often unknown to the average Internet user. For instance, Google's Safe Browsing (GSB) service offers a dedicated web portal[1] for users to report potential phishing websites. However, it is unlikely that most regular users are even aware of this portal, let alone use it. Despite this, all prior studies on web-based phishing reporting systems [25, 26, 43, 44] rely on this portal to evaluate GSB's resilience against evasive phishing attacks.

In this paper, we aim to address this gap by concentrating our research solely on *email-based* phishing reporting systems. To the best of our knowledge, email-based phishing detectors and phishing-reporting systems have never been systematically studied before. Email remains the most dominant vector for cyberattacks, accounting for over 91% of all attacks [8], with more than a trillion phishing emails sent worldwide each year [8]. Given such a large prevalence of phishing emails, it is crucial to study the phishing reporting systems that are integrated into popular email services and gauge their resilience to potential evasion attacks. These built-in systems are likely far more familiar and accessible to regular users compared to the isolated, web-based systems examined in prior research. For example, Gmail provides an intuitive phishing reporting mechanism, as shown in Fig. 1.

Objective. With this paper, we aim to answer two main questions: (Q1) How do email services handle the phishing reports that they receive from users? (Q2) Can attackers abuse email-based phishing reports to fingerprint phishing detectors and cloak their malicious infrastructure from them, achieving long-lasting phishing campaigns?

Key idea: weaponizing email tracking. Our key idea is based on a threat vector that is traditionally associated with web privacy, that of *email tracking* [29]. Email tracking enables senders to monitor if and when an email has been opened, and in some cases, where it was accessed from. This is achieved by leveraging the fact that most email clients support HTML-based emails with embedded remote objects, such as images. The loading of these images serves as a side channel to deduce the act of email opening.

[1] https://safebrowsing.google.com/safebrowsing/report_general/

Ever since Google made the loading of images in emails the default option in 2013 [47], pixel tracking has become a more widely feasible way for marketers to track the success of their campaigns via email open rates [1]. In this paper, we explore the idea of using similar techniques but from a security perspective to launch evasive attacks against the anti-phishing systems hosted by email services. In particular, we evaluate the use of email-tracking techniques to not only target the end users but all subsystems employed by email services, both before and after an email is flagged for phishing. Thus, we show that a vector that was associated with web privacy is "doubly dangerous" since attackers can weaponize these vectors for better phishing attacks.

Our experimental results reveal an alarming finding: the phishing reporting systems of all major email services we tested—serving over 2 billion users—can be profiled and evaded, including two of the most widely used email platforms, Google Mail and Microsoft Outlook. This conclusion stems from a systematic measurement experiment we performed in which we sent thousands of emails with custom tracking vectors over a period of 44 days. Our experiment showed that all studied email services employed separate subsystems for activities such as prefetching, proxying, and phishing inspection. All these subsystems, unfortunately, displayed *differential behavior* that makes them vulnerable to "email open" tracking. This differential behavior manifested in a variety of ways, such as *identifiable HTTP headers, limited network address diversity, and differential loading* when initiating requests of objects embedded in our tracking emails. Importantly, the fact that the phishing inspection subsystems can be tracked directly points to the affected services being vulnerable to evasive phishing attacks. We confirm this with the help of end-to-end simulation phishing experiments that we designed and implemented.
We summarize our key contributions below.

1. We systematically improve the array of candidate vectors that can be used for tracking emails beyond the commonly used pixel-tracking techniques. Using these improved tracking vectors, we confirm that most popular email services (7 out of 8) are prone to "email open tracking" in their default settings (§ 3).

2. Weaponizing these vectors, we perform a systematic measurement study on 4 popular email services that offer a phishing reporting system. Our study confirms the existence of various subsystems, all of which (including phishing reporting systems) can be uniquely identified by attackers via multiple network signals during the loading of email objects (§ 4).

3. We confirm the validity and stability of our findings by designing and conducting a small-scale, end-to-end simulated phishing attack experiment. The results show the longevity of phishing sites powered by our evasive attack approach in comparison to baseline attack sites, which all get immediately blocked (§ 5).

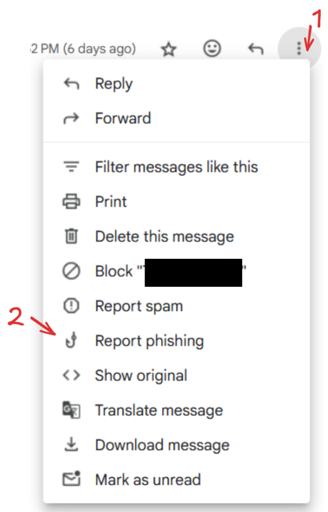

Figure 1: *A Gmail user needs to make two clicks as shown to report an opened email as "phishing."*

We also discuss multiple countermeasures available for email services (§ 6). While some measures present usability issues and need a nuanced look at the security and usability tradeoffs, others are more straightforward with no foreseeable usability side effects. Thus, we expect this latter variety of recommendations to be quickly adopted by major email service providers as a result of our disclosure process.

2 Attack overview

In this section, we first establish the threat model for the evasive phishing attacks we examined in this study. We then follow it up with a high-level sketch of the attack mechanism that we propose and evaluate.

2.1 Threat model

2.1.1 The phishing attacker

In this paper, we examine the scenario of a phishing attacker aiming to lure potential victims into visiting their phishing website. As previously discussed, this attacker chooses to use email as a delivery vehicle for their attacks owing to its popularity [8]. This means they will craft a link to their phishing website and share it with the victims through a persuasively written email. Considering that phishing is an attack with very low success rates, the attacker will need to send a large number of phishing emails in total.

As phishing sites that get reported to anti-phishing entities typically have lifespans of only a few hours [25, 43], the

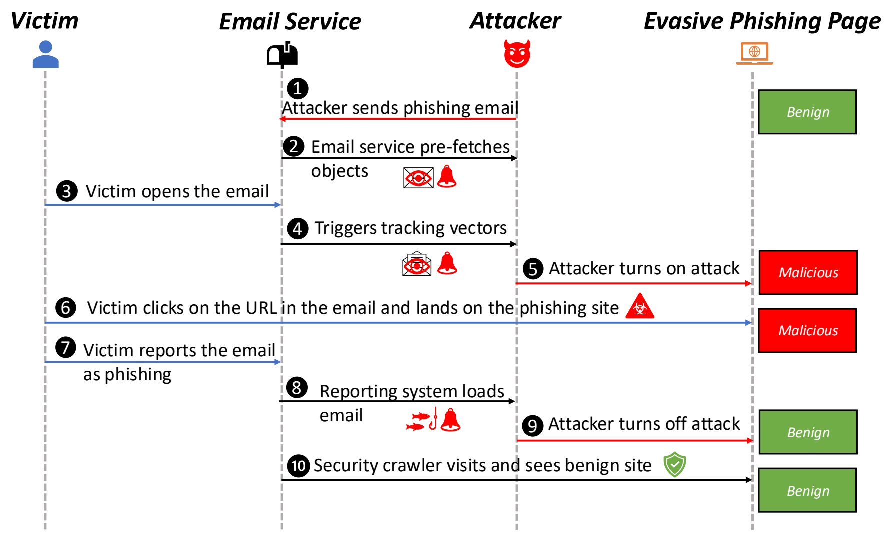
Victim
Email Service
Attacker
Evasive Phishing Page
1 Attacker sends phishing email
Benign
2 Email service pre-fetches objects
3 Victim opens the email
4 Triggers tracking vectors
5 Attacker turns on attack
Malicious
6 Victim clicks on the URL in the email and lands on the phishing site
Malicious
7 Victim reports the email as phishing
8 Reporting system loads email
9 Attacker turns off attack
Benign
10 Security crawler visits and sees benign site
Benign

Figure 2: *An example evasive phishing attack scenario where the victim recognizes the attack and reports it*

attacker seeks to increase the longevity of their phishing websites. To do this, the attacker sets up their phishing website in such a way that each receiver has a unique URL link to their site. We also assume that the attacker has the ability to turn each of these links into a phishing or benign mode. Thus, the same website might simultaneously carry some links that are benign while several others that are malicious.

2.1.2 Phishing victims

In this study, we assume that the victims being *targeted* by the attacker have retained the default configuration in their email clients regarding remote image loading. As we will discuss in § 3, modern email clients typically allow images to load by default, and we consider this setting to be applicable to the targeted victims. This assumption is supported by prior research, which indicates that the majority of users (>70%) do not alter default privacy settings [4] and often prefer images to load automatically [3].

However, this does not imply that the attacker avoids sending phishing emails to *untargeted users* who disable remote image loading. Instead, the attacker still sends them emails but ensures that the attack remains inactive for such users. To do this, the attacker configures the URL shared with untargeted users to operate in the aforementioned "benign" mode.

It is also worth noting that users who actively modify their default image-loading settings are generally more security-conscious and, therefore, less likely to fall victim to phishing attacks. By keeping shared URLs in benign mode for such users, the attacker reduces the likelihood of their phishing sites being flagged and blocked due to reports from untargeted recipients. Finally, in § 7, we will discuss how our experimental results support the inclusion of even these more security-savvy users in our threat model.

Ultimately, all users (whether targeted or not) are assumed to receive their emails and open them at any time. While some users open the links and fall for the phishing attack, other users will report the email as "phishing" to email services. The main objective of the attacker is to attempt to keep their phishing site active for as long as possible despite multiple phishing reports from users (whether targeted or not).

2.2 Evasive attack mechanism

With the threat model established, we now outline a high-level overview of our proposed evasive phishing attack mechanism using an example scenario illustrated in Fig. 2. The figure depicts an attacker dynamically managing an evasive phishing webpage (i.e., a URL). The attacker initially sets the webpage to a benign mode and sends a phishing email embedded with tracking vectors to a targeted user ❶. Upon receiving the email, the email service prefetches objects (such as images) embedded in it, which triggers an alert to the attacker ❷. However, since these requests are identified as coming from an email service's data prefetching mechanism (based on the tracking vector information), the attacker keeps the phishing webpage in benign mode.

When the victim opens the email ❸, it generates a characteristic tracking signal that alerts the attacker ❹. This time, recognizing that the event is from a potential victim rather than a prefetching service, the attacker switches the webpage to phishing mode ❺. The victim then clicks the URL and lands on the phishing page ❻. Upon realizing the nature of the site, the victim reports the email as "phishing" ❼. This prompts the email service's phishing detection system to load the email's objects, generating yet another tracking signal to the attacker ❽. In response, the attacker immediately reverts the phishing webpage to benign mode ❾, effectively hiding it from subsequent visits by phishing detection crawlers ❿.

In this scenario, it is crucial to note that the tracking signals revealing the victim's actions, such as opening the email ❹ and reporting it ❻, play a pivotal role in enabling the attacker to execute evasive changes to the phishing webpage. In the following sections, we will explore how these tracking signals can be realized across major email services that support phishing reporting.

Alternative scenarios. While the above scenario highlights one specific attack sequence, the attacker's ability to monitor all key email events, such as service prefetching, user email opening, and phishing reporting, makes this evasion approach adaptable to various alternative scenarios. For instance, consider an untargeted user who reports the email. Since such users' email clients do not generate "email open" signals, step ❺ is bypassed, and the phishing webpage remains in benign mode during all subsequent visits by security crawlers.

3 Tracking vectors in modern email services

From our previous section, it is clear that we need to investigate the efficacy of email tracking vectors for attack purposes. For this, we first study how feasible it is to track the opening of emails in the default client settings. Specifically, we expect to capture the signals that emanate during a potential email prefetching phase (❷ in Fig. 2) and a subsequent email opening phase (❸, ❹ in Fig. 2.) The key idea that enables this tracking is the fact that modern email clients support many HTML and CSS features that can be exploited to track the rendering of emails. A sender can include an external resource in an email, such as an image with a unique tracking URL. When the recipient interacts with the email, the email client might initiate an HTTP request to the attacker's server to load that resource. The tracking server receives this request and can leverage the unique identifier in the resource to attribute it to a specific recipient. We refer to the HTML and CSS features as **tracking vectors** and the specific action of initiating the associated HTTP requests as **triggering** the tracking vectors.

3.1 New e-mail tracking vectors

Previous works [29, 30, 31] have identified pixel tracking via `<img>` as the primary method used to track email recipients. Englehardt et al. [29] evaluated different email clients to assess their defense against pixel tracking. Since modern email clients support multimedia in HTML and CSS, we leveraged this feature towards email tracking to look beyond simple pixel tracking via `<img>` tags. To this end, we systematically curated a list of all non-deprecated HTML tags that support the loading of external resources by reading through the official MDN documentation. This resulted in a list of 15 HTML tags. Among them, ultimately only 11 HTML tags were useful as tracking vectors[2] in at least one of the three attack stages we discuss in the paper. For brevity, we only discuss these useful vectors in detail in Table 13 in the Appendix. The table lists the names of these tracking vectors, along with their description, usage examples, and links to documentation.

One interesting example among these useful HTML tags is the `<input>` tag whose `type` attribute can be set to the value `"image"` to create a submit button that displays an image instead of a standard button[3]. Similarly, we also found `<object>`, `<picture>`, `<svg>`, `<audio>`, `<video>` tags to be useful for attack purposes, as we will see later.

Next, to complement these pure HTML features we also devised three tracking vectors using remote CSS property features (also listed and discussed in Table 13). While CSS has been recently studied for fingerprinting purposes in richer environments such as full-fledged web pages [54], our objective here is to only measure its utility when used inside HTML email content for email tracking and phishing attack evasion purposes. For example, the CSS rule `@font-face` supports a method to source fonts from a remote server, which we found can be abused for email tracking.

3.2 Analyzed e-mail services

We selected a list of 15 email service providers as potential candidates for our study based on popularity and accessibility [48, 49]. We then performed a feasibility check on each of them. As one of our requirements for a later experiment (§ 4) is to create about 10 email accounts, we disregarded any services that did not permit us to do this. Five email services were regional in nature and did not permit us to create accounts from our jurisdiction (e.g., 163.com, mail.ru). After this process, we were left with eight popular email services that we consider in this study including highly popular services like Gmail, Outlook, Yahoo, and Proton. Table 1 presents these services along with their estimated user counts, collectively covering billions of users.

3.3 Methodology

We built a simple automated system capable of composing and sending emails consisting of these HTML tracking vectors. Each tracking vector is assigned a unique tracking URL that enables the system to distinguish the tracking vectors and identify the recipients across multiple emails. We used this system to measure the effectiveness of all the tracking vectors we curated against the eight email services. Our goal was to find the vectors that get triggered during the prefetching and email opening phases of an email's life cycle.

[2]The 11 HTML tags resulted in 14 tracking vectors with <img> and `<object>` tags being leveraged in multiple forms as tracking vectors. More details are in Table 13.

[3]Interestingly, this vector was triggered in AOL and Yahoo even when image loading was disabled in the email client settings

Email Services	Total Users (million)
Gmail	1800.0
Outlook	400.0
Yahoo	230.0
Proton Mail	100.0
Mail.com	14.6
GMX	11.0
Tuta Mail	10.0
AOL	1.5

Table 1: *Total number of estimated users for the eight email providers we consider in our study [2, 6, 7, 32]*

We created one account for each email service to conduct this experiment. We then registered a `.com` domain to serve the requests for the email tracking vectors. Before sending the tracking emails, each email account was warmed up by sending a couple of emails back and forth between the eight accounts to create a trust relationship. We then sent three test emails with all tracking vectors embedded and then opened them manually. This process was conducted on both web (Windows 11 desktop) and Android App platforms (Xiaomi Poco F2 Pro). We made sure to retain all the accounts in their default configuration settings in order to align with our threat model as previously discussed (§ 2.1).

3.4 Results

Table 2 and Table 12 (in the Appendix) show the behavior of email clients against different tracking vectors on the web and Android platforms, respectively. The columns of these tables show all tracking vectors that were successfully triggered on at least one email service. Each tick in the cells indicates a trigger event of a tracking vector either during email prefetching (prior to email opening) or an email opening event. Double ticks indicate both events. Cells are colored in red whenever a tracking vector got activated during the email opening event. Interestingly, the results show that, under their default settings, *all email providers except Tuta Mail leak at least one email open signal to the tracking server,* accounting for more than 2 billion users, indicating a grave privacy threat.

In addition to simple pixel tracking, both tables show that various additional vectors such as image submit, picture, font, audio, video, and CSS inclusion are all effective for tracking email open activities amongst various email clients. It is useful to note the difference in the behavior of email clients across these tracking vectors. For example, in the case of Proton Mail in Android (Table 12), regular pixel tracking with `<img>` does not leak the email open signal. However, using image tracking with `<picture>` or in paragraph background using CSS gives an email open signal.

Our results also show that three of the eight services we examined (Gmail, Proton, Outlook) tend to prefetch certain types of objects. Interestingly, we also observed that some vectors (such as the "SVG" vector in Proton) are triggered during the prefetching phase but not during the subsequent email opening phase. We surmise that this indicates a "caching" feature implemented by these services for specific objects.

We also examined the IP address of the requests triggered via tracking vectors during email opening to assess the potential leakage of the end user's address. As shown in Tables 2 and 12, three services: Outlook, GMX, and Mail.com all reveal the client's IP address on both platforms in their default settings. We also found that Tuta Mail also leaks a user's IP address whenever they override settings to enable image loading. In contrast, services such as Gmail, Yahoo, and Proton route user requests via proxies for added privacy.

4 Evaluation of email reporting systems

Building on our findings of how the server-side prefetch and user-side email open activities are revealed to attackers, we now investigate what happens when a user reports an opened email as "phishing" to the service. For this, we perform an end-to-end controlled experiment where we send neutral-text emails embedded with various tracking vectors to our own accounts. We then open and report a subset of these emails as "phishing" with the goal of answering both the questions we pose in § 1 as part of our quest to study the email phishing reporting infrastructure of popular services.

4.1 Experimental setup

Our setup consisted of a *Sending* module responsible for composing and sending emails with tracking vectors. The full list of vectors used by this module is listed in Table 13. Additionally, each email was also fitted with a unique URL linked to a honey website that we controlled. All the emails were reported using our *Reporting* module that uses GUI automation.

4.1.1 Email services

For this experiment, we selected the email providers based on whether they provided a built-in dedicated email phishing reporting button. Among the eight popular email services we evaluated (Table 1), four—Gmail, Outlook, Proton Mail, and Tuta Mail—provide users with a dedicated button for reporting phishing emails. The remaining services—Yahoo, AOL, GMX, and mail.com—lack a dedicated button to report an email as phishing. Rather, for these providers, the recommended approach for reporting emails is to mark the phishing email as spam[4]. Hence, for our experiment, we omit these email providers. Still, we note that the four remaining email services we cover in our experiment, namely, Gmail, Outlook,

[4]When the sender is from the same platform as the receiver, these services provide an email address to forward the phishing email to them. However, this is not applicable to us as we send emails cross-platform.

Vector / Service	bg image	image submit	img src	img src large	para bg image	picture	font	iframe	object html	object vid	SVG	IP Leak
Gmail	×	✓✓	✓✓	✓✓	✓✓	✓✓	×	×	×	×	×	×
Outlook	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	×	✓
Yahoo	×	✓	✓	✓	✓	✓	×	×	×	×	×	×
Proton	✓	×	✓✓	✓✓	✓✓	✓✓	✓	×	×	×	✓	×
Mail.com	×	×	✓	✓	✓	✓	×	×	×	×	×	✓
GMX	×	×	✓	✓	✓	✓	×	×	×	×	×	✓
Tuta	×	×	×	×	×	×	×	×	×	×	×	×*
AOL	×	✓	✓	✓	✓	✓	×	×	×	×	×	×

Table 2: *Tracking vectors for prefetch and email open stages of various services on the web platform. Each ✓ represents the vector getting triggered during one of the stages. All cases in which the vector was triggered during email open stage are in red indicating serious security implications. Cases in which vectors are activated only during prefetch stage are in yellow (with a single ✓) while all others are in green. ✓✓ denotes the vectors triggered at both stages.* $\times^*$ *denotes the IP address of the client leaking when overriding default settings to allow image loading.*

Proton Mail, and Tuta Mail, altogether account for more than 2.3 billion users worldwide as listed in Table 1.

4.1.2 Email accounts

We manually created both the receiving and the sending accounts. The experiment was designed to send emails to the receiving accounts in a cross-platform manner. We deliberately chose this approach to minimize the information available to the receiving email service regarding the sending accounts, thereby reducing the likelihood of the sending accounts being flagged as suspicious or malicious. By minimizing the association between the sender and the receiver, we tried to create a more realistic simulation and mitigate any potential bias.

We separated all the receiving accounts belonging to the same service by creating a dedicated Google Chrome profile for each account (10 total profiles). We took this approach to isolate each account from others within the same service. Additionally, we assigned a specific VPN server location to each profile and accessed the accounts exclusively from these chosen locations. This was done to ensure diversity in the IP addresses among the email accounts and create a setup mimicking a real-world email delivery and phishing reporting.

4.1.3 Sending module

The *Sending* module was responsible for composing each email, embedding the tracking vectors, and delivering the emails to the receiving accounts. For crafting emails, we made a collection of 10 randomly selected books from Project Gutenberg[5]. The content of each email was generated by extracting a few random sentences from the book collection. Each email was assigned a randomized value (between 0 and 1) which is used by the *Reporting* module to select emails for reporting to introduce randomness and non-determinism in the email reporting process. Each tracking vector URL in the email had a unique random string embedded in it, which made it possible to track each vector across all emails. In addition to the tracking vectors, each email contained a unique URL link to a honey website.

[5] `https://www.gutenberg.org/`

4.1.4 Honey websites

We set up multiple honey sites with the aim of evaluating the behavior of security crawlers of the email services that may visit the URLs embedded in the email. For this, each honey site recorded the IP address and the HTTP request headers for each visit. Additionally, we also included fingerprinting code in the website to profile the crawlers that visit the URLs included in the emails. Specifically, we added canvas and JS-based font fingerprinting code in the honey websites. Canvas fingerprinting [41] computes a fingerprint as a cryptographic hash of a hidden image drawn on the webpage. Font fingerprinting [42] uses JavaScript to detect the list of fonts installed in the client and generates a cryptographic hash of the font list, serving as a fingerprint. Canvas and JS-based font fingerprinting have been used effectively in fingerprinting and evading security crawlers [25].

4.1.5 Reporting module

The *Reporting* module was used to report emails received by each receiving account using OS-based GUI automation. For this, the module utilized PyAutoGUI for input control and Tesseract for optical character recognition (OCR) on a Windows VM. This module was responsible for signing into each account using a dedicated VPN location to ensure IP address consistency. To avoid triggering CAPTCHAs, we deliberately avoided using browser automation tools such as Selenium. We believe that this use of OS-based GUI automation paired with the steps to main IP address consistency for account access helped in successfully avoiding CAPTCHAs during our experiments. The reporting module can randomly report specific emails based on the random weight value assigned for each email. This way, our experimental pipeline was capable of reporting a chosen fraction of random emails as phishing without any temporal biases.

4.2 Deployment

We conducted this experiment using 40 receiving email accounts across four email services. We manually created 10 receiving accounts for each email service. Additionally, we created 20 sending accounts, five each for Google, Yahoo, Outlook, and AOL. Before the experiment, all receiving and sending accounts were warmed up and also subscribed to newsletters to make them appear as legitimate, active users to email service providers. The sending accounts were warmed up by exchanging at least four emails per account. The receiving accounts exchanged at least four emails per account. The emails were exchanged exclusively between cross-platform services. To separate accounts belonging to the same platform, we created 10 Google Chrome profiles for the receiving accounts and five profiles for the sending accounts.

We used the sending module to compose the emails with neutral content to minimize spam. It is important to note that for this experiment, we are concerned about studying the email reporting infrastructure and not the spam filtration system. To this effect, any emails that ended up in a recipient's junk folder were manually moved to the inbox without opening the email. This also does not affect our attack scenario discussed at § 2 and § 3 because the attack is not activated until opened by the recipient. We also assume that the attacker has the capability and resources to craft an email that ends up in the recipient's inbox rather than the spam folder. For the honey sites, we registered 20 domains under the `.com` TLD. Each URL consisted of a domain and a randomly generated string, such as `https://domain/random-str`, which ensures that it is unlikely to be visited by unintended visitors.

We sent a total of 1,997 emails across 40 email accounts in a cross-platform manner. We used the reporting module to report, at random, approximately 70% of the emails received by continuously increasing the threshold value for reporting. We completed this experiment over a period of 44 days.

4.3 Results

Table 3 provides a comprehensive breakdown of the total number of emails received and phishing reports made for each email service. Upon careful analysis of the data collected during this experiment, we found that some email tracking vectors were triggered shortly after email delivery, even without opening the email. Furthermore, we saw that numerous tracking vectors were triggered when the email was opened and, crucially, after the email was reported as phishing. These different triggering requests and events allowed us to identify distinct systems associated with the email services. These systems were distinguishable by their *distinct identifiable HTTP headers* as well as *differential loading* of tracking vectors. This latter differential loading refers to the observed phenomenon of a characteristic subset of tracking vectors getting triggered whenever an email is processed by one of the service's internal email systems. This creates a discernible pattern of tracking vectors for some systems that we refer to as a *tracking vector pattern* (Table 4, 5, 6). The distinct HTTP headers paired with these tracking vector patterns make the email systems susceptible to tracking. This behavior was observed for Gmail, Outlook, and Proton Mail, but not for Tuta Mail. Because Tuta Mail did not trigger any tracking vectors at all, we exclude it from the findings.

Based on these observations of distinct behaviors tied to specific events, we categorize and present our results into three categories: *Prefetching Resources*, *Email Open*, and *Email Reporting*. The systems involved in each of these categories exhibit discernible behavior through either distinguishable HTTP headers or a unique tracking vector pattern or both.

Service	Received	Opened	Not Opened	Reported	Not Reported
Gmail	497	398	99	339	158
Outlook	500	469	31	344	156
Proton	500	399	101	349	151
TutaMail	500	447	53	349	151

Table 3: *Total number of emails received, opened, unopened and reported for each email service*

System Prefetch (*p*) Open (*o*) Report (*r*)	Pattern %	image submit	img src	img src large	para bg image	picture
G1 - *p*	92.6%	✓	✓	✓	✓	✓
G2 - *o*	100%	✓	✓	✓	✓	✓
G3 - *r*	99.2%	×	✓	✓	×	✓
G4 - *r*	96.1%	×	✓	✓	×	✓

Table 4: *Dominant patterns for Gmail showing triggering (✓) and non-triggering (×) of various email tracking vectors*

4.3.1 Prefetching resources

Our data showed that, sometimes, Gmail, Outlook, and Proton triggered some tracking vectors even before the emails were opened. This prefetching of resources is consistent with our findings in the earlier section (§ 3). Table 7 shows the total

System Prefetch (*p*) Open (*o*) Report (*r*)	Pattern %	bg image	image submit	img src	img src large	para bg image	picture	a href	address	CSS	font	iframe	object html	object image	object vid	SVG
O1 - *p*	100%	×	×	×	×	×	×	×	×	×	×	✓	×	×	×	×
O2 - *p*	97.1%	×	×	×	×	×	×	×	×	×	×	×	✓	×	✓	×
O3 - *o*	91.3%	✓	✓*	✓	✓	✓	✓	×	×	×	×	×	×	×	×	×
O4 - *r*	76.5%	✓	×	✓	✓	✓	✓	×	×	✓*	×	✓	✓	✓*	✓	✓*
O5 - *r*	64.5%	✓	×	×	×	×	×	✓*	✓*	×	✓*	×	×	×	×	×

Table 5: *Dominant patterns for Outlook showing triggering (✓) and non-triggering (×) of various email tracking vectors.* * *denotes the tracking vector is unique and persistent in all (100%) of the system's visits*

System Prefetch (*p*) Open (*o*) Report (*r*)	Pattern %	bg image	image submit	img src	img src large	para bg image	picture	font	SVG
P1 - *p*	100%	✓	×	✓	✓	✓	✓	✓*	✓
P2 - *o*	95.2%	×	×	✓	✓	✓	✓	×	×
P3 - *r*	70%	✓	✓*	✓	✓	✓	✓	×	✓

Table 6: *Dominant patterns for Proton showing triggering (✓) and non-triggering (×) of various email tracking vectors.* * *denotes the tracking vector is unique and persistent in all (100%) of the system's visits*

number of prefetched emails along with the request headers of the respective systems prefetching the resources. The identfiable patterns in the headers are shown in bold indicating the parts that make them vulnerable to tracking.

We noticed that a single Gmail system (**G1**) was associated with prefetching resources in 309 out of 497 emails. This system was easily identifiable due to its `referer` header, `"http://mail.google.com/"` (note: the use of HTTP is not a typographical error). With Outlook, we were able to divide the requests into two subsystems based on slight differences in their header values (**O1** and **O2**). However, similar to Gmail, both of them had characteristically unique headers (Table 7).

In addition to HTTP headers, we also noticed a tracking vector pattern for these prefetching systems. We found that for certain systems, these tracking vector patterns were always consistent, while in some cases, there was minor variability with some vectors either not getting triggered or triggered multiple times. For example, we see that for the Outlook system **O1**, the tracking vector pattern is always the same (Table 5), but for Gmail's **G1**, the tracking vector pattern can vary slightly, so we report the most dominant pattern. We found the most dominant tracking vector pattern in 92.5% of the prefetched emails (Table 4). Similarly, we identified a unique tracking vector pattern for all prefetched Outlook emails, as illustrated in Table 5 for **O1**. For Proton Mail, the system (**P1**) responsible for prefetching consistently displayed a tracking vector pattern (refer to Table 6) across all prefetched emails.

We are not sure of the exact reasons behind this prefetching and can only speculate on the intended functionality of these black box systems. The prefetching of the resources could be an optimization measure implemented to enhance the performance of the email clients or a privacy measure against email tracking to give a false open signal. However, due to the combination of distinct headers and unique pattern of tracking vectors triggered by these systems while prefetching resources, these systems are susceptible to tracking.

Summary. We find that Gmail, Outlook, and Proton Mail each prefetch tracking vectors before email opens—identifiable by unique HTTP headers (Gmail, Outlook) and consistent vector patterns (Proton, Outlook)—making these "prefetch" subsystems all vulnerable to tracking.

4.3.2 Email open

We observed distinct email open signals from services. As in § 3, we found that some tracking vectors get re-triggered during email open despite being triggered earlier during prefetch. Table 8 shows the HTTP headers of the systems associated with the email open action while the tracking vector patterns for each system are detailed in Tables 4, 5, and 6.

The Gmail system (**G2**) associated with email opens had a distinct `User-Agent` header that was easily identifiable by the string: `GoogleImageProxy`. **G2** also showed a consistent vector pattern across all the opened emails. However, since this pattern resembled that of **G1**, we cannot rely solely on it to differentiate between prefetching and email open events. However, Tables 7 and 8 show that **G1** and **G2** have very different HTTP headers providing a clear distinction.

For Outlook, the `User-Agent` and the user's IP address are leaked due to the lack of a proxy. Email prefetching can be easily distinguished from an email open by examining the request IP or the unique HTTP headers (Table 8). For prefetching resources, the request IP address belonged to a Microsoft Autonomous System (AS), while we observed the recipient's IP address on email opens. Moreover, the tracking

Service	ID	Prefetched	Identifiable Headers
Gmail	**G1**	309 (61.8%)	`"user-agent":["Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36` **`Edge/12.246 Mozilla/5.0`**`"],"referer":[`**`"http://mail.google.com/"`**`]`
Outlook	**O1**	215 (43.0%)	**`"sec-fetch-dest":["iframe"],`**`"user-agent":["Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)` **`HeadlessChrome/128.0.6613.18 Safari/537.36"`**`],`
	O2	209 (41.8%)	**`"sec-fetch-dest":["object"],`**`"user-agent":["Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)` **`HeadlessChrome/128.0.6613.18 Safari/537.36"`**`]`
Proton	**P1**	75 (15.0%)	`"user-agent":["Mozilla/5.0` **`(X11; Linux x86_64)`** `AppleWebKit/537.36 (KHTML, like Gecko)` **`Chrome/125.0.0.0`** `Safari/537.36"]`

Table 7: *Identifiable parts of headers in HTTP requests seen during email prefetching across various services. The third column shows both the absolute number of prefetched emails and as a percentage of the total emails received.*

Service	ID	Proxy	Identifiable Headers
Gmail	**G2**	✓	`"user-agent":[`**`"Mozilla/5.0 (Windows NT 5.1; rv:11.0) Gecko Firefox/11.0 (via ggpht.com GoogleImageProxy)`**`"]`
Outlook	**O3**	×	`"referer":[`**`"https://outlook.live.com/"`**`],"user-agent": readers's browser UA`
Proton	**P2**	✓	`"user-agent":["Mozilla/5.0` **`(X11; Linux x86_64)`** `AppleWebKit/537.36 (KHTML, like Gecko)` **`Chrome/125.0.0.0`** `Safari/537.36"]`

Table 8: *Identifiable parts of headers in HTTP requests seen when users open their emails across various services. The third column shows whether the HTTP requests are being routed via a proxy.*

vectors requested for prefetching were different from those of email open, as shown in Table 5. We can thus rely on the tracking vectors as well as the HTTP headers to differentiate between Outlook's email prefetching and email open events.

For Proton, the HTTP headers were identical for prefetching (**P1**) and email open (**P2**), which was via its proxy. However, we found that the reliability of the tracking vector patterns helped distinguish between an email open and prefetching, thanks to our diverse collection of tracking vectors. We observed that certain tracking vectors are exclusively requested during prefetching and never during the email open, as shown in Table 6. For example, prefetching (**P1**) can be easily distinguished from an email open (**P2**), because tracking vectors such as `bg image`, `font`, and `SVG` are absent for **P2**. Note that, to differentiate between **P1** and **P2**, it suffices to check for just one unique tracking vector rather than matching the entire pattern, which further simplifies the process.

The combination of unique HTTP headers and tracking vector patterns exhibited by these systems involved in prefetching and opening emails enables an attacker to accurately determine when an email has been opened. This has significant implications for the attack scenario that we discussed in § 2.

Summary. We find that each service triggers a distinct "email open" signal—identifiable by HTTP headers (Gmail identified by its User-Agent, Outlook via unique HTTP headers/IP), and unique tracking vectors (Proton)—making email open events reliably distinguishable from prefetching.

4.3.3 Email reporting

Given that we can already reliably differentiate between prefetching and an email open, the ability to track email phishing reports would be key for our proposed attack (§ 2). For this, we observed additional HTTP requests being made by all services after the emails were reported as phishing. This indicates that the emails were rendered again for examination by the systems involved in the reporting infrastructure. Table 9 shows the HTTP request headers and the percentage of reported emails that triggered a post-report request. Overall, we found that each system involved in handling email reporting is distinct from systems responsible for prefetching resources and opening emails, as shown by variations in their HTTP headers and unique tracking vector patterns.

We identified two systems (denoted **G3** and **G4**) with distinct `User-Agent` headers for Gmail (Table 9). The **G3** system, which included the string `Gmail-content-sampling` in its HTTP headers, was observed in 245 out of the 339 emails that were reported. Crucially, this HTTP header was absent in all 158 non-reported emails, clearly indicating some level of email inspection conducted by Gmail's email reporting infrastructure. The **G4** system, with a distinct HTTP header having the string `GoogleOther`, was found in 51 emails out of the 339 reported emails. We note that **G4** is likely not a security-related crawler, as it was intended to reduce the crawl capacity of `Googlebot`[50]. Both systems, **G3** and **G4**, can be differentiated from an email opened by the recipient due

Service	ID	Reported	Identifiable Headers
Gmail	G3	245/339 (72.3%)	`"user-agent":["Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36 **Edge/12.246 Gmail-content-sampling**"]`
	G4	51/339 (15.0%)	`"user-agent":["**GoogleOther**"]`
Outlook	O4	342/344 (99.4%)	`"user-agent":["Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) **Chrome/109.0.0.0** Safari/537.36"]`
	O5	342/344 (99.4%)	`**"sec-fetch-dest": ["document"]**, "user-agent": ["Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/<125/126/127> Safari/537.36"]`
Proton	P3	10/349 (2.9%)	`"referer":["**https://phishing.protontech.ch/**"],"user-agent":["Mozilla/5.0 (*; *) AppleWebKit/537.36 (*, like Gecko) Chrome/129.0.0.0 Safari/537.36"]`

Table 9: *Identifiable parts of headers in HTTP requests seen after reporting a subset of emails as "phishing" across systems of various services. The third column shows the number of emails for which requests from the specific anti-phishing system were seen and the total number of emails reported to that service.*

to the difference in the `User-Agent` string and the difference in the pattern of tracking vectors (Table 4).

For Outlook, we observed 342 out of 344 reported emails had requests from a system (**O4**) with a specific `109.0.0.0` Chrome version number in the `User-Agent` string. The system **O5** was also associated with the same 342 reported Outlook emails and had a distinct HTTP header as detailed in Table 9. Both **O4** and **O5** exhibit unique tracking vectors as seen in Table 5 that are exclusively triggered by these systems, rendering these reporting systems susceptible to tracking. With Proton, we noticed **P3**'s requests as containing a distinct URL in their `Referer` header: `https://phishing.protontech.ch`, in 10 out of 349 reported emails. Along with this HTTP header, we also found that tracking vector patterns lead to the identification of Proton's anti-phishing systems, as `image submit` vector was uniquely triggered only by **P3** as shown in Table 6.

Fig. 3 shows the time taken to get tracking vector requests from the email reporting systems of Gmail, Proton, and Outlook after an email was reported. We see that reporting systems from Outlook and Gmail are quick to respond to phishing reports. Specifically, **O4**, **O5** and **G3** systems triggered these tracking vectors within a minute of the user report in most cases. In contrast, Proton's reporting system was significantly slower in comparison. It responded within several minutes for fewer than 40% of these emails but took several hours to days to trigger tracking vectors in the remaining 60%. This substantial delay in Proton's system means attackers can reliably evade detection by simply using timing-based cloaking attacks, where phishing content goes benign within a few minutes of email open.

Summary. We find that each reporting service gives a unique "post-report" signal—Gmail's via its `User-Agent` header, Outlook's via its unique HTTP headers, and Proton's via its `Referer` header as well as a unique tracking vector pattern (`image submit` vector)—making email phishing reporting activities of end users apparent to attackers prior to any security actions pursued by email services.

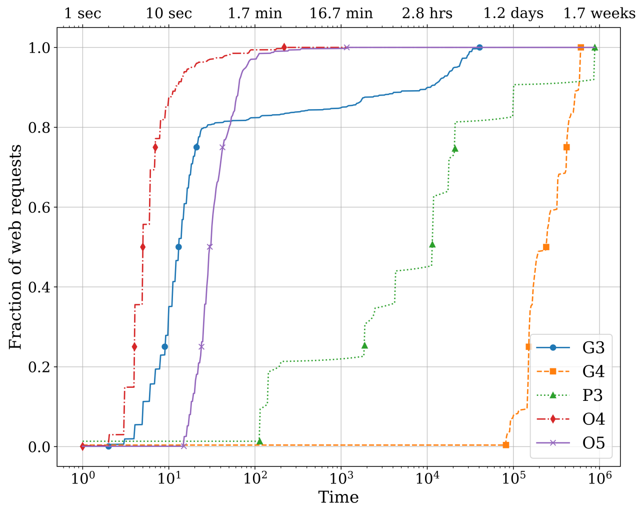

Figure 3: *Time distribution of tracking signal requests after reporting emails for Gmail, Outlook and Proton Mail. X-axis is presented in two scales for readability purposes (lower X-axis is in seconds).*

4.3.4 Honey website visits

We observed visits to the honey URLs included in the emails that were reported to Outlook. The emails that were reported to Gmail and Proton did not trigger any visits to the honey websites. We received a total of 317 visits from 344 reported Outlook emails. Each of these visits to the honey URL immediately followed the corresponding email report event. We observed only 16 unique IP addresses across these 317 visits in our dataset, all of which belonged to the Microsoft AS. In the case of JS-based fingerprints, most of Outlook's crawlers emanated only a single unique Canvas and JS-based font fingerprint (214/317 visits). In the remaining visits (103/317), we did not receive any fingerprints during the Outlook crawlers' visits indicating a non-standard browser which in itself is anomalous behavior. Thus, the low diversity in these fingerprints, fingerprint anomaly, and limited network address diversity can all be leveraged to successfully evade Outlook's phishing reporting crawlers as described in prior work [25].

4.4 Implications

Overall, we find that the presence of identifiable HTTP headers, limited network address diversity, and differential loading of the tracking vectors by the email systems make it possible to identify and distinguish each event (prefetching resources, email opening, and email reporting) and each system. This information gained from this experiment further solidifies our idea of an evasive phishing attack leveraging email tracking vectors discussed in § 2. It is clear that the opening of the emails can be tracked reliably under the default settings employed by these email clients. Furthermore, the email reporting systems for these services also leak information through the email tracking vectors and HTTP request headers. Such leaked information can be used to identify the loading of emails by these security crawlers and devise a cloaking-based phishing attack. An attacker can activate a phishing attack when the recipient opens the email. If the email is reported, the tracking vectors give the attacker a signal, in which case, the phishing payload can be deactivated.

5 Simulation of an end-to-end attack

In this section, we present a confirmation experiment simulating an end-to-end phishing attack to reinforce the implications of our findings in § 4.3 and demonstrate the practicality of our attack discussed in § 2 and § 4.4. We show the attack's ability to evade detection despite aggressive email reporting by leveraging the email tracking vectors covered in § 4. To demonstrate the effectiveness of these vectors in launching successful phishing campaigns, we built evasive phishing websites powered by intelligence from email tracking vectors and measured their lifespan against browser blocklists.

5.1 Setup

Our setup consisted of the *Sending* module (§ 4.1.3), the Reporting module (§ 4.1.5), a collection of simulated phishing websites, and a *Monitoring* module. We sent emails consisting of unique URLs to our phishing websites across email accounts of three services: Gmail, Proton, and Outlook. For each email, we visited the embedded URL and then reported the email as phishing. We reused the receiving accounts described in § 4.1.2, but created new sending accounts, which we warmed up before using them for the experiment.

5.1.1 Sending module

We made minor modifications to the *Sending* module described in § 4.1.3. Specifically, to simulate a real-world phishing email, we modified the email subject and content to reflect a phishing context rather than the neutral tone used previously. The modified content informed recipients of a failed transaction and urged them to visit the embedded URL to verify their PayPal transaction history. This URL in the email directed users to a simulated phishing site.

5.1.2 Reporting module

We used the Reporting module from § 4.1.5 with two key modifications. First, we disabled Google Chrome's client-side *Safe Browsing*[6] feature, which is enabled by default to provide client-side protection against phishing and other malicious content. Chrome also sends full URLs and bits of page content to Google if a site appears suspicious. Therefore, to ensure any blocking to the phishing sites was solely by the security crawlers of the email services—and not by the client-side phishing protection—we disabled the Safe Browsing feature during reporting emails.

Second, we modified the Reporting module to click on the embedded URL and capture a screenshot of the phishing page before reporting the email. We do this to simulate a potential real-world victim who is likely to click on the URL and land on the phishing site. This step also allows us to confirm that the phishing sites are live and serving phishing content to victims. Additionally, we also included this step to account for any potential behavioral differences triggered by clicking on the embedded URL, which were not observed in § 4.3.

5.1.3 Simulated phishing sites

For this experiment, we use two varieties of simulated phishing websites: smart evasive sites and baseline sites following the approach used in prior work [25, 44]. The baseline sites are simple sites that show phishing content to any visitor whenever the URL embedded in an email is accessed. If the URL is not associated with an email, benign content is served. We use the baseline sites to validate our experiment as realistic by demonstrating that these sites get successfully blocked as expected through our email reporting process.

The smart sites, in contrast, leverage the intelligence from the email tracking vectors to dynamically decide whether to serve phishing or benign content. The smart sites use the evasion logic discussed in § 2 (Fig. 2). Specifically, we leverage the findings from § 4.3.2 to track the opening of specific emails. Only after this particular event occurs for a given email will a subsequent visit to the URL embedded in the email, trigger the smart site to serve phishing content. Similarly, based on insights from § 4.3.3, we can detect when an

[6] https://safebrowsing.google.com/

email has been reported, at which point the smart site permanently switches to serving benign content for the associated URL. We note that it was not necessary to utilize our findings from § 4.3.4 to evade Outlook's phishing crawler visits, as the email open and reporting signals alone were sufficient for the success of our experiments.

5.1.4 Monitoring module

Our *Monitoring* module periodically checks whether phishing sites are active or have been blocked by web browsers. We use Google Chrome and Microsoft Edge to check if the sites have been blocked by their respective GSB and SmartScreen blocklists, which together cover the majority of web users. Prior work has used a similar approach to evaluate the lifetime of phishing sites [25, 43, 44]. Our module automates this process by using GUI automation to load the URLs in both browsers with default settings, capturing a screenshot and timestamp for each visit. We manually verify the screenshots to determine if a site is still active or has been blocked.

5.2 Deployment

We conducted this experiment 41 days after the conclusion of the previous experiment described in § 4. This intentional gap between the two experiments was to assess the stability of the tracking vectors and HTTP headers across the email services observed previously. We registered six `.com` domains—three for baseline sites and three for smart sites. The phishing payloads consisted of lookalike login pages for PayPal and Bank of America, as well as a replica of a Microsoft pop-up scam. For benign content, we redirected the visitor to PayPal's official support page, consistent with the email's content of PayPal payment issues. We removed the `Referer` header in the redirect request to avoid leaking our URLs to PayPal. Any request to URLs not associated with any email was served a generic "site under construction" message.

The *Sending* module sent 45 emails per email service across three baseline domains, with each email containing a unique URL. These emails were randomly sent across 10 receiving accounts per service, totaling 135 emails for the baseline sites. For the smart sites, we sent 120 emails per service, each containing a unique URL, resulting in 360 emails in total. The *Reporting* module reported all the received emails as "phishing". We conducted our experiment over a 24-day period in January 2025.

5.3 Results

Our results show that the three baseline sites were blocked quickly, while the smart websites remained active for 24 days until the end of our experiments. The first block to the baseline sites took only around 4 hours and 30 minutes after the first email report. The baseline sites received a total of 114 visits from the crawlers of the email services. In contrast, our smart websites were not blocked despite getting a total of 275 visits from the crawlers belonging to the email services. Table 10 shows the detailed results of reporting emails for the baseline sites, while Table 11 shows the results of reporting emails for the smart sites.

Service	**# Reports**	**# Visits**	**# URLs**	**# Unique IP**	**# Unique AS**
Gmail	45	50	21 (46.6%)	39	25
Outlook	45	60	41 (91.1%)	16	1
Proton	45	4	1 (2.2%)	4	1

Table 10: *Phishing experiment: traffic for baseline sites*

Service	**# Reports**	**# Visits**	**# URLs**	**# Unique IP**	**# Unique AS**	**# Open Loads**	**# Report Loads**
Gmail	120	153	60 (50.0%)	110	45	120	120
Outlook	120	119	119 (99.2%)	15	1	120	120
Proton	120	3	1 (0.8%)	3	2	0	0

Table 11: *Phishing experiment: traffic for smart evasive sites*

Despite starting the experiment 41 days after the conclusion of our previous experiment, we observed consistent behavior across systems—identical headers and tracking vector patterns, as observed in § 4. Furthermore, we observed similar behavior with Outlook's crawlers visiting the phishing sites as seen in § 4.3.4. For the baseline sites, 15 of the 16 visiting IPs, and for smart sites, all 15 Outlook IPs, had already been encountered in the previous experiment. 131 out of 179 requests to the phishing sites had one unique Canvas fingerprint and JS-based font fingerprint which was the same as seen previously. The remaining requests anomalously did not provide any fingerprints just as before. Additionally, all the visiting crawlers belonged to the single AS, `MICROSOFT-CORP-MSN-AS-BLOCK, US`. These results, consistent with our previous findings show a critical lack of fingerprint, IP, and AS diversity in Outlook's crawler infrastructure shows their highly susceptible to alternative fingerprinting and evasive attacks aimed directly at their web crawlers [25].

In addition to the URL visits after email reporting, we observed that Gmail uses a click-time link protection[7] feature, where we noticed Gmail crawlers proactively visiting embedded URLs shortly *after* a user clicks on them—regardless of

[7] https://support.google.com/mail/answer/10173182

whether the email is reported. While these crawlers exhibited diversity in AS and IP addresses, as shown in Table 10 and Table 11, they failed to execute any fingerprinting code, which can be leveraged to evade these crawlers. However, our attack model (§ 2) does not rely on this anomaly for evasion. Instead, we serve the phishing payload to the recipient only once, triggered by the recipient's initial "email open" signal. Upon clicking the link, the user is shown the phishing page. This action, however, turns the payload benign for subsequent visits. If the same recipient revisits the link, they must reopen the email and retrigger the "email open" signal to activate the malicious payload again. This mechanism ensures that Gmail's security crawlers never encounter the phishing payload while the intended recipient does.

6 Recommended countermeasures

As discussed in § 4, our attacks primarily rely on the ability to infer the prefetching, email opening, and email reporting activities via tracking signals. So, by blurring these signals, one can potentially suppress these attacks as we discuss below.

Email reporting signals. The most dangerous signal is arguably the email reporting signals that are triggered whenever a reporting system internally loads a candidate phishing email's objects (§ 4.3.3). In our attack, this signal immediately deactivates the phishing content, thus hiding the web page permanently from the reporting system. To address this, email service operators should consider the following measures:

1. Operators can completely suppress the email reporting signals by not loading any remote objects after receiving the reports from users. While this will mitigate our attack, it may cause collateral damage as other systems might rely on the missed image content (e.g., brand logos [36]) to determine the overall phishing intention of the email.
2. Even if the above solution were implemented, the attacker can still employ a 'temporal defusion" logic to automatically deactivate their attack payload after a predefined time window following the opening of an email. Given that a typical user spends only 9 seconds on average with an email [5], this will give a very short window for the phishing reporting systems to act promptly and capture the live phishing payload. We thus recommend email services to respond quickly upon receiving email reports. Current data seems to suggest that only Google and Outlook act quickly, while Proton is slower (Fig. 3).
3. Caching the email objects during prefetching instead of reloading them after reporting is another defensive option. But this demands enormous storage capacity as all remote objects in emails need to be stored for a long time to accommodate users who might not immediately open and report their emails.
4. Instead of suppressing email report signals, a reporting system could simply emulate the service's corresponding "email open" signals (see Table 8, for example). In particular, there is no valid reason for anti-phishing systems of Gmail and Proton to "announce" their presence in their request headers (see Table 9). However, this approach will still be insufficient in the case of targeted users, as the "email open" signal will now be triggered twice, which may raise suspicion and prompt evasive action by the attacker. In contrast, for untargeted users, this approach is effective as the email load only happens a single time, thus leaving the phishing page exposed in its activated state increasing the likelihood of detection.
5. Email service providers should also strongly consider homogenizing the network behavior of their reporting, prefetching, and proxying systems. This includes their source network addresses as well as the lists of specific remote objects that they load, i.e. the triggered tracking vector patterns (see Tables 4, 5, 6). From an attacker's perspective, this will make all email-related activities (either prefetching or reporting) look similar thus confusing them about whether the subsequent site visit is from a victim or an anti-phishing system.

Email open signals. In this paper, we showed that the email tracking signals that were traditionally considered a threat to privacy also pose a security threat when exploited for phishing attacks. Specifically, the fact that most email clients support email open tracking by default (§ 3) enables attackers to activate phishing pages in a just-in-time manner for a limited duration. To prevent this, email clients can reverse their default setting of automatically loading images. This will be "doubly defensive", offering both security and privacy benefits to users. However, it might introduce usability issues, as users will have a reduced default quality of their email service due to blocked images. An alternative as discussed earlier, would be for the email service to cache all email objects during prefetching and thus suppress all email open signals, unfortunately requiring significant storage resources. It is also important to note that reliably blocking all remote objects is challenging and prone to errors due to the increasing complexity of the web HTML, as shown in prior work [46].

Another mitigation strategy is for services to emit a non-deterministic number of "email open" signals. This prevents an attacker from reliably distinguishing a legitimate "email open" by a user from one generated by the service. This disrupts the attacker's ability to reliably activate the phishing payload. Note that a single, deterministic signal is insufficient, as the attacker can simply choose to ignore it. Thus, a non-deterministic number of "email open" signals is important here. However, an attacker can still choose to treat the first "email open" signal as genuine and proceed with the attack. In such a scenario, the conversion rate for the attacker will go down, but a limited number of victims may still be affected.

In summary, we recommend that carefully homogenizing email activity signals—such as HTTP headers, supported HTTP objects and crawling infrastructure across all stages of processing, combined with non-deterministic pre-rendering, offers a viable defense against our evasion attacks while balancing the storage and usability constraints.

7 Discussion

Alternative attacks. We now discuss some alternative evasion attacks compared to ours. Single-use URLs that deliver the phishing payload only once can be one such alternative. While this approach can prevent subsequent detection, it is still vulnerable to premature detection. Specifically, if a cautious target victim reports the email without opening the link, the link will now be prone to the anti-phishing crawler and will be detected. In our attack on the other hand, email phishing reporting activity deactivates the phishing payload making it elusive to the anti-phishing crawlers. Gmail employs a "click-time link protection" feature that proactively visits embedded URLs after a user clicks on them. This URL crawling happens regardless of whether the email has been reported, making it unreliable to use the "email report" signal alone to deactivate the phishing content. Nevertheless, as discussed in § 5.3, this preemptive crawling does not render our attacks ineffective due to the activation and deactivation of phishing payload after the victim's email open and visit events respectively. As discussed in the previous section, a careful homogenization of email activity signals combined with a non-deterministic of pre-rendering activities provides the best chance to defend against our proposed evasion attacks.

Limitations. One limitation of our study is the potential miss on capturing the full range of behaviors in our email accounts that real-world users may experience. To minimize this, we made sure that each account for an email service was connected through a VPN from a unique geographic location (§ 4.1.2). Another limitation of our study arises from the inherent nature of these email services as black box systems. While our goal is to study the behavior of email services towards emails that are reported as phishing, we accept that we cannot for certain know if other services like spam filtering or other heuristics, interfere with this behavior. Since the email services we study are black boxes, we may not always be entirely correct in our explanation of certain events. For example, in the case of Proton, only 10 out of the 349 reported emails triggered the tracking vectors (§ 4.3.3). While we observe and accurately report these numbers, we can only speculate that this may be occurring due to heavy email sampling at Proton Mail's end. Similarly, our reasoning for resource prefetching in some emails being prefetched is a valid argument rather than a definitive conclusion (§ 4.3.1). However, we stress that regardless of the arguments, the end effect observed is still the same in terms of threats to privacy and security.

Responsible disclosure. We followed a responsible disclosure protocol and submitted tailored vulnerability reports to each affected email provider, detailing the issues presented in this paper along with recommended mitigations. We received acknowledgements from all these services (Google, Proton, and Microsoft). Further, Google confirmed opening a bug report based on our disclosure and indicated that fixes are in progress. Our disclosure report was also awarded with a Google Vulnerability Reward of US$ 2,337.00. It was placed in the category of "abuse-related methodologies" with "Medium impact" and "Medium exploitation likelihood" indicating the real-world relevance of our research findings. Proton's security team informed us that they implemented some fixes to remedy the attack, most notably suppressing external resource loading for phishing reporting systems. They also confirmed that they were exploring anti-fingerprinting techniques and improving their image caching. Microsoft is still reviewing the issue with its security team as of the time of compiling this manuscript. In addition to the disclosure report, we also provided an option to embargo our paper until the conference to give the service providers ample time window to fix the issues that we identified in our disclosure report. One service provider requested this, citing the need for additional time to implement all the necessary fixes due to the "complexity of the case".

We also submitted a disclosure report to other service providers that we evaluated for privacy against tracking vectors (§ 3), but did not evaluate their email reporting systems (§ 4). Our disclosure report described our findings and the potential for attack against their phishing reporting systems, along with recommendations to counteract them.

8 Related work

Email privacy. Prior work has studied the implications of email tracking across various email clients. Englehardt et al. [29] identified widespread privacy risks in email clients through pixel tracking as well as leaking information to third-parties through headers, e.g., `Referer`. It is encouraging that some email clients have adopted the use of a proxy after the implications of email tracking were highlighted in previous work [29, 55]. While we analyze different email clients for their privacy, our primary objective is to demonstrate how poor privacy features of these email clients can be exploited as a security vulnerability to launch highly effective phishing attacks. Previous work [29, 30, 31, 55] has identified pixel tracking as the primary method used to track email recipients. We have expanded the email tracking to additional vectors along with pixel tracking. Some of these tracking vectors e.g., `<link>` to include CSS, `@font-face` have been evaluated in prior works in the context of CSS-based fingerprinting [35, 53, 54] while our use case is for email open tracking.

Email reporting. The phishing reporting ecosystem has been comprehensively evaluated for smishing, vishing, and phishing email attacks in previous work [52]. Additionally, in their work, Sun et al.[52] reported phishing sites through publicly available channels and not built-in dedicated function e.g., "Report phishing Button" in Gmail. In our work, we focus on the reporting of phishing emails on email services using the built-in dedicated button to report phishing emails to understand their effectiveness. Moreover, we are concerned about the behavior of these security crawlers employed by the email reporting systems from a privacy point of view.

Fingerprinting and cloaking attacks. Prior work has employed cloaking techniques against security crawlers that inspect websites using controlled phishing sites [25, 28, 40, 43, 44, 56]. These studies focused on limitation of crawlers that visit the websites for inspections whose visits were solicited via dedicated web sites setup to receiving phishing URL reports [25, 43]. However, our research does not focus on such URL reports but rather evaluates the reporting made through dedicated buttons provided by each email service. Our evasion techniques proposed are solely based on email tracking vectors that thus far have only been deemed to have privacy implications for email clients.

The advantage of our attack over previously seen fingerprinting-based cloaking techniques lies in its use of email interaction signals that makes for a more powerful evasion mechanism in the context of email-based phishing attacks. Prior attacks, such as PhishPrint [25], cloak content to specific crawlers with known fingerprints. This means that the effectiveness of this approach is critically dependent on maintaining accurate fingerprint profiles for a large and evolving set of crawlers that can visit the phishing site, as seen previously in [25]. Any previously unseen crawler will be served the phishing content revealing the nature of the phishing site. In contrast, our attacks rely on characteristic "email open" tracking signals to activate the phishing site content only after the email has been opened/rendered by the receiver. Thus, any unknown crawler that stumbles upon the site will not see the phishing content, leading to more elusive phishing sites. Additionally, as we saw in our work (§ 4), the diversity of the email subsystems is very limited compared to the web security crawlers [25] that inspect the web pages. Hence, it is easier to maintain a relatively short list of identifiable headers and tracking vectors to keep track of email interaction signals, abusing them to launch elusive phishing content.

9 Conclusion

We performed the first study focused on real-world phishing email reporting systems. For this, we weaponized e-mail tracking, a well-known privacy issue, in a new security-oriented context of evasive phishing attacks. In this attack setting, we evaluated the resilience of popular email services and found that all of them were vulnerable to evasive phishing attacks affecting more than 2 billion people. We confirm these weaknesses in the form of end-to-end simulated phishing experiments, which demonstrate the ability to create long-lasting phishing campaigns using our attack mechanism. Finally, we discuss several countermeasures to help ameliorate this issue.

Ethics considerations

We approached this research with careful and deliberate attention to ethical responsibility, aiming to minimize any potential harm while evaluating the email services. Although we did not obtain prior consent from the email providers involved in our experiments, we carefully considered this decision. We determined that the practical challenges of identifying and reaching out to the appropriate point of contact within such large organizations (email providers) authorized to provide consent to our work made obtaining consent infeasible. Due to this, we designed our methodology along the lines of previous works [25, 28, 40, 43, 44, 56].

Our experiments involved sending a total of 1,997 emails for the study of email reporting infrastructure (§ 4) over several days, and 495 emails over five days to evaluate the response to phishing reports (§ 5). We argue that both the volume and the rate of these emails are negligible in comparison to the daily scale handled by these services. The total number of phishing reports submitted across the entire study was 1,876. While we acknowledge that each email reported by us imposes a waste of time and resources for these email services, we argue that the total number of email reports (1,876) is very small in comparison to the large number of email reports that these services receive daily. For context, Gmail blocks around 100 million phishing emails every day [17], making our total of 1,876 email reports (across all services) negligible.

To further evaluate the ethical implications of our approach, we consulted with the authors of related prior work [25] while planning our experiments, who shared their experience with us. They described how the product lead from Google Safe Browsing encouraged their team to continue sending phishing report URLs, justifying that the volume of phishing reports sent out by them had a negligible impact on Google, which receives an astronomical number of phishing reports daily.

Additionally, during the account-creation phase of our experiments, some of our accounts were flagged and blocked by ProtonMail. We reached out to them and clarified that the accounts were created as part of our research project. ProtonMail acknowledged the need for our study and reactivated our accounts, indicating they were happy to let us continue with our study. In light of the feedback, statistics, and the overall setup of our experiments, we concluded that neither our emails nor our reports negatively affected any of these services. Instead, the findings from our paper can help strengthen these services against sophisticated evasions that attackers could already be silently exploiting the identified email-open signals in the wild. Furthermore, prior work in evaluating web-based phishing reporting systems [25, 26, 28, 43, 44] has also followed a similar procedure of submitting small-scale false phishing reports to investigate the security of real-world anti-phishing systems.

On the website front, the phishing sites were simulated and non-functional and thus not harmful to real users in the unlikely event that they would somehow stumble upon them. The URLs were not shared outside of the reported emails, both for research results' fidelity and ethical purposes. The email accounts were not accessible by anyone except the authors, and these accounts were not used to communicate with real users. For creating email accounts, we used personal phone numbers whenever required. Further, we have also clearly communicated the details of our measurements, including the number of false reports we made to these services in our disclosure reports to these email services.

Open science

To comply with the open science policy, we have made the code from our work publicly available[8]. We would like to note that some of the data and code can be abused by malicious parties against the email services to launch their own phishing attacks. This includes the code used in launching phishing attacks, dataset with the IP Addresses of the crawlers that are used by these email services. To understand how prior work [25] approached the disclosure of similar data, we approached Acharya and Vadrevu, who shared that they received a request from the anti-phishing scanners to not make the data of their hosts publicly available. In a similar vein, we have not made this forensic data publicly available. However, we have made this data available on a restrictive basis to bona fide researchers via Zenodo[9].

Acknowledgments

We would like to thank the reviewers and the shepherd for their valuable feedback to this paper. This work was supported by the Army Research Office (ARO) under grant W911NF-24-1-0051 and the National Science Foundation (NSF) under grants CNS-2211575, CNS-2126654, CNS-1941617, and CNS-2422035.

References

[1] Email tracking in gmail: How google improved it? https://web.archive.org/web/20211101000000*/https://openedornot.com/email-tracking-in-gmail.

[2] Mail.com shouldn't stay master of its domains. https://web.archive.org/web/20230827111311/https://www.forbes.com/2000/06/02/feat.html?sh=9e2f79a3730b, 2013.

[3] Gmail data analysis reveals image blocking affects 43 https://www.litmus.com/blog/gmail-data-analysis-reveals-image-blocking-affects-43-of-emails, 2014.

[4] Norton cyber safety insights report: Special release – online creeping. https://www.nortonlifelock.com/us/en/newsroom/press-kits/2022-norton-cyber-safety-insights-report-special-release-online-creeping/, 2022.

[5] Trends in email engagement. https://www.litmus.com/blog/trends-in-email-engagement, 2022.

[6] Celebrate with us: Tutanota reaches 10 million users! https://web.archive.org/web/20241008100450/https://tuta.com/blog/10-million-users, 2023.

[7] Top 10 most popular email providers in the world. https://web.archive.org/web/20241002154816/https://www.alltopeverything.com/most-popular-email-providers/, 2024.

[8] Top phishing statistics for 2025: Latest figures and trends. https://web.archive.org/web/20250115050350/https://www.stationx.net/phishing-statistics/, 2024.

[9] <a>: The anchor element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/a, 2025.

[10] <address>: The contact address element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/address, 2025.

[11] <audio>: The embed audio element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/audio, 2025.

[12] background-image - css: Cascading style sheets: Mdn. https://developer.mozilla.org/en-US/docs/Web/CSS/background-image, 2025.

[13] @font-face - css: Cascading style sheets: Mdn. https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face, 2025.

[14] <iframe>: The inline frame element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/iframe, 2025.

[15] <img>: The image embed element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/img, 2025.

[16] <input>: The html input element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/input, 2025.

[17] The latest 2025 phishing statistics. https://web.archive.org/web/20250117122906/https://aag-i

[8]Public repository: https://doi.org/10.5281/zenodo.15612284

[9]Restricted repository: https://doi.org/10.5281/zenodo.15612193

t.com/the-latest-phishing-statistics/, 2025.

[18] <link>: The external resource link element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/link, 2025.

[19] <object>: The external object element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/object, 2025.

[20] <p>: The paragraph element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/p, 2025.

[21] <picture>: The picture element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/picture, 2025.

[22] <svg> - svg: Scalable vector graphics: Mdn. https://developer.mozilla.org/en-US/docs/Web/SVG/Reference/Element/svg, 2025.

[23] <video>: The video embed element - html: Hypertext markup language: Mdn. https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/video, 2025.

[24] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. Visualphishnet: Zero-day phishing website detection by visual similarity. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[25] Bhupendra Acharya and Phani Vadrevu. Phishprint: Evading phishing detection crawlers by prior profiling. In *30th USENIX Security Symposium*, 2021.

[26] Bhupendra Acharya and Phani Vadrevu. A human in every APE: delineating and evaluating the human analysis systems of anti-phishing entities. In *Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA*, 2022.

[27] Hugo L. J. Bijmans, Tim M. Booij, Anneke Schwedersky, Aria Nedgabat, and Rolf van Wegberg. Catching phishers by their bait: Investigating the dutch phishing landscape through phishing kit detection. In *30th USENIX Security Symposium*, 2021.

[28] Yoonjung Choi, Woonghee Lee, and Junbeom Hur. Phishinwebview: Analysis of anti-phishing entities in mobile apps with webview targeted phishing. In *Proceedings of the ACM on Web Conference 2024, WWW*, 2024.

[29] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. I never signed up for this! privacy implications of email tracking. *Proc. Priv. Enhancing Technol.*, 2018.

[30] Johannes Haupt, Benedict Bender, Benjamin Fabian, and Stefan Lessmann. Robust identification of email tracking: A machine learning approach. *Eur. J. Oper. Res.*, 2018.

[31] Hang Hu, Peng Peng, and Gang Wang. Characterizing pixel tracking through the lens of disposable email services. In *2019 IEEE Symposium on Security and Privacy, SP*, 2019.

[32] Steven Knight. Most popular email providers by number of users. https://web.archive.org/web/20240720004207/https://www.sellcell.com/blog/most-popular-email-provider-by-number-of-users/, 2023.

[33] Brian Kondracki, Babak Amin Azad, Oleksii Starov, and Nick Nikiforakis. Catching transparent phish: Analyzing and detecting MITM phishing toolkits. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[34] Yuexin Li, Chengyu Huang, Shumin Deng, Mei Lin Lock, Tri Cao, Nay Oo, Hoon Wei Lim, and Bryan Hooi. Knowphish: Large language models meet multimodal knowledge graphs for enhancing reference-based phishing detection. In *33rd USENIX Security Symposium*, 2024.

[35] Xu Lin, Frederico Araujo, Teryl Taylor, Jiyong Jang, and Jason Polakis. Fashion faux pas: Implicit stylistic fingerprints for bypassing browsers' anti-fingerprinting defenses. In *44th IEEE Symposium on Security and Privacy, SP*, 2023.

[36] Yun Lin, Ruofan Liu, Dinil Mon Divakaran, Jun Yang Ng, Qing Zhou Chan, Yiwen Lu, Yuxuan Si, Fan Zhang, and Jin Song Dong. Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages. In *30th USENIX Security Symposium*, 2021.

[37] Ruofan Liu, Yun Lin, Xiwen Teoh, Gongshen Liu, Zhiyong Huang, and Jin Song Dong. Less defined knowledge and more true alarms: Reference-based phishing detection without a pre-defined reference list. In *33rd USENIX Security Symposium*, 2024.

[38] Ruofan Liu, Yun Lin, Xianglin Yang, Siang Hwee Ng, Dinil Mon Divakaran, and Jin Song Dong. Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. In *31st USENIX Security Symposium*, 2022.

[39] Ruofan Liu, Yun Lin, Yifan Zhang, Penn Han Lee, and Jin Song Dong. Knowledge expansion and counterfactual interaction for reference-based phishing detection. In *32nd USENIX Security Symposium*, 2023.

[40] Sourena Maroofi, Maciej Korczyński, and Andrzej Duda. Are you human? resilience of phishing detection to evasion techniques based on human verification. In *Proceedings of the ACM Internet Measurement Conference*, 2020.

[41] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP*, 2012.

[42] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Gio-

vanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy, SP*, 2013.

[43] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Kevin Tyers. Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists. In *2019 IEEE Symposium on Security and Privacy, SP*, 2019.

[44] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. Phishtime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *29th USENIX Security Symposium*, 2020.

[45] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *29th USENIX Security Symposium*, 2020.

[46] Damian Poddebniak, Christian Dresen, Jens Müller, Fabian Ising, Sebastian Schinzel, Simon Friedberger, Juraj Somorovsky, and Jörg Schwenk. Efail: Breaking S/MIME and openpgp email encryption using exfiltration channels. In *27th USENIX Security Symposium*, 2018.

[47] John Rae-Grant. Official gmail blog: Image now showing. https://gmail.googleblog.com/2013/12/images-now-showing.html, 2013.

[48] Mohammed Saqib. 15 biggest email service and account providers. https://web.archive.org/web/20250109144038/https://finance.yahoo.com/news/15-biggest-email-account-providers-184704151.html, 2022.

[49] Mohammed Saqib. 5 biggest email service and account providers. https://web.archive.org/web/20221201213010/https://www.insidermonkey.com/blog/5-biggest-email-service-and-account-providers-1090933/, 2022.

[50] Barry Schwartz. Google launches a new crawler named googleother. https://searchengineland.com/google-launches-new-googlebot-named-googleother-395827, 2023.

[51] Karthika Subramani, William Melicher, Oleksii Starov, Phani Vadrevu, and Roberto Perdisci. Phishinpatterns: measuring elicited user interactions at scale on phishing websites. In *Proceedings of the 22nd ACM Internet Measurement Conference, IMC*, 2022.

[52] Zhibo Sun, Faris Bugra Kokulu, Penghui Zhang, Adam Oest, Gianluca Stringhini, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. From victims to defenders: An exploration of the phishing attack reporting ecosystem. In *The 27th International Symposium on Research in Attacks, Intrusions and Defenses, RAID*, 2024.

[53] Naoki Takei, Takamichi Saito, Ko Takasu, and Tomotaka Yamada. Web browser fingerprinting using only cascading style sheets. In *10th International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA*, 2015.

[54] Leon Trampert, Daniel Weber, Lukas Gerlach, Christian Rossow, and Michael Schwarz. Cascading spy sheets: Exploiting the complexity of modern CSS for email and browser fingerprinting. In *32nd Annual Network and Distributed System Security Symposium, NDSS*, 2025.

[55] Haitao Xu, Shuai Hao, Alparslan Sari, and Haining Wang. Privacy risk assessment on email tracking. In *2018 IEEE Conference on Computer Communications, INFOCOM*, 2018.

[56] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kapravelos, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing. *IEEE Secur. Priv.*, 2022.

[57] Penghui Zhang, Zhibo Sun, Sukwha Kyung, Hans Walter Behrens, Zion Leonahenahe Basque, Haehyun Cho, Adam Oest, Ruoyu Wang, Tiffany Bao, Yan Shoshitaishvili, Gail-Joon Ahn, and Adam Doupé. I'm spartacus, no, i'm SPARTACUS: proactively protecting users from phishing by intentionally triggering cloaking behavior. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2022.

A Figures

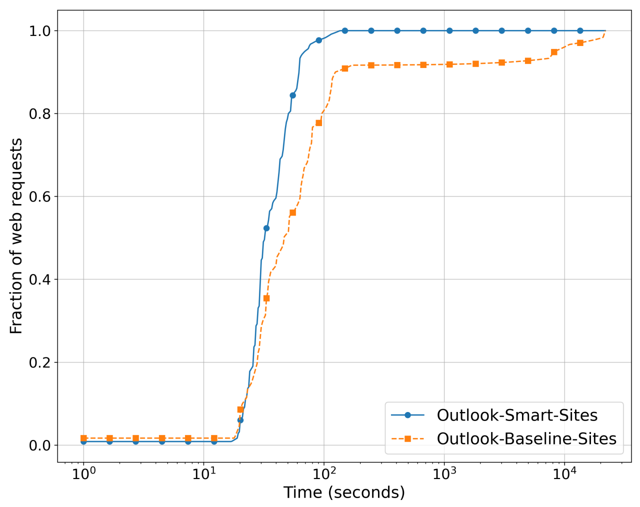

Figure 4: *Time distribution of the Outlook visits after email report*

Vector Service	bg image	image submit	img src	img src large	para bg image	picture	audio	CSS	font	SVG	video	IP Leak
Gmail	×	✓	✓	✓	✓	✓	×	×	×	×	×	×
Outlook	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓	✓
Yahoo	×	✓	✓	✓	✓	✓	×	×	×	×	×	×
Proton	✓	×	×	×	✓	✓	×	×	✓	✓	×	×
Mail.com	✓	×	✓	✓	✓	✓	×	✓	×	×	×	✓
GMX	✓	×	✓	✓	✓	✓	×	✓	×	×	×	✓
Tuta Mail	×	×	×	×	×	×	×	×	×	×	×	×*
AOL	×	✓	✓	✓	✓	✓	×	×	×	×	×	×

Table 12: *Tracking vectors for prefetch and email open stages of various services on the Android platform. Each ✓ represents the vector getting triggered during one of the stages. All cases in which the vector was triggered during email open stage are in red indicating serious security implications. Cases in which vectors are activated only during prefetch stage are in yellow (with a single ✓) while all others are in green.* $\times^*$ *denotes the IP address of the client leaking when overriding default settings to allow image loading.*

Name	Trigger Action	Description	Tracking Vector Usage Example	Ref.
List of all non-deprecated HTML tags that refer to remote URLs that can be leveraged for e-mail tracking				
a href	CLICK	Used to create hyperlinks to navigate to other pages	`<a href="TRACKING_URL"></a>`	[9]
address tag	CLICK	Used to define contact information	`<address><a href="TRACKING_URL"></a></address>`	[10]
audio	RENDER	Used to embed audio content	`<audio controls preload="auto" autoplay><source src="TRACKING_URL/song.ogg" type="audio/ogg"></audio>`	[11]
CSS	RENDER	Used to link external CSS file	`<link rel="stylesheet" href="TRACKING_URL/app.css">`	[18]
iframe	RENDER	Used to embed another HTML document within the current page	`<iframe src="TRACKING_URL" height="2px" width="2px"></iframe>`	[14]
image submit	RENDER	Used to add an external image in form submission button	`<input type="image" src="TRACKING_URL/logo.png" width="2px" height="2px">`	[16]
img src	RENDER	Used to add an external image to the HTML email from specified source URL (1x1 size). This specific size is monitored separately to identify any distinct behaviors associated with tracking pixels, which are commonly used in emails [31].	`<img src="TRACKING_URL/logo_1x1.png"/>`	[15]
img src large	RENDER	Used to add an external image to the HTML email from specified source URL (100x100 size)	`<img src="TRACKING_URL/logo_100x100.png" />`	[15]
object html	RENDER	Used to embed HTML documents or multimedia	`<object data="TRACKING_URL" width="2px" height="2px"></object>`	[19]
object image	RENDER	Used as an alternative method to embed images	`<object data="TRACKING_URL/img.png" width="2px" height="2px"></object>`	[19]
object vid	RENDER	Used as an alternative method to embed videos	`<object data="TRACKING_URL/video.mp4" width="2px" height="2px"></object>`	[19]
picture	RENDER	Used to set multiple image sources	`<picture><source srcset="TRACKING_URL/img1.png" media="(orientation: portrait)" /><img src="TRACKING_URL/img2.png" alt="" /></picture>`	[21]
svg inline	RENDER	Used to embed vector graphics in HTML	`<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink = "http://www.w3.org/1999/xlink"> <image xlink:href="TRACKING_URL/img.svg"/> </svg>`	[22]
video	RENDER	Used to include video content	`<video preload="metadata" width="2" height="2" controls autoplay><source src="TRACKING_URL" type="video/mp4"></video>`	[23]
Sample CSS properties we used that refer to remote URLs that can be leveraged for e-mail tracking				
bg image (CSS)	RENDER	Used to set background image	`body {background-image: url("TRACKING\_URL/img.png");}`	[12]
font (CSS)	RENDER	Used to include external fonts	`@font-face{font-family: "myFont"; src: url("TRACKING_URL");}`	[13]
para bg image (CSS)	RENDER	Used to set background image for a paragraph using CSS	`<p style="background-image:url( "TRACKING_URL/bg.png" );"> </p>`	[20]

Table 13: *All HTML and CSS tracking vectors that can track pre-fetching, opening, and reporting stages of emails in the eight email services we studied in this project. The "Trigger Action" column describes what the e-mail receiving client (i.e. prefetch email server, end user, or phishing report system) has to do to trigger the tracking associated tracking vector: either a CLICK on the link in the email or a simple RENDER of the e-mail's content.* `TRACKING_URL` *refers to a unique URL path created and controlled by the attacker used for tracking email activities.*