Characterizing the Phishing Landscape via JavaScript APIs

Anonymous authors

Abstract

In 2024, the Anti-Phishing Work Group identified over one million phishing pages. Phishers achieve this scale by using phishing kits — ready-to-deploy phishing websites — to rapidly deploy phishing campaigns with specific data exfiltration, evasion, or mimicry techniques. In contrast, researchers and defenders continue to fight phishing page-by-page and rely on manual analysis to recognize static features for kit identification, researchers better characterize trends in the face of mass-deployed kits, and defenders to tie kits to specific malicious actors.

This paper aims to aid researchers and analysts by automatically differentiating groups of phishing pages based on the underlying kit, automating a previously manual process, and enabling us to measure how popular different client-side techniques are across kits. For kit detection, our system has an accuracy of 95% on a ground-truth dataset of 548 kits families deployed across 4,562 phishing URLs. On an unlabeled dataset, we leverage the complexity of 744,903 phishing pages' JavaScript logic to group them into *15,063* clusters, annotating the clusters with what phishing techniques they employ and exploring which techniques are universal, widespread across kits, or kit-specific. We find that UI interactivity and basic fingerprinting (User-agent, Cookies, and Referrer) are the most universal techniques, present in 90% and 80% of the clusters, respectively. On the other hand, mouse detection via the browser's mouse API is among the rarest behaviors, despite being used in a deployment of a 7-year-old open-source phishing kit. We show that the sophistication of the client-side code and mass-deployment of kits can be measured by observing their browser API traces, and leverage traces obtained from 744,903 to categorize different client-side techniques.

1 Introduction

Web-based phishing attacks, where a webpage, through mimicking or urgency, tricks the user into submitting personal information to an attacker, have been increasing for the last 5 years [13]. Phishing attacks can have high-profile targets, like the NGOs and government workers targeted in 2021 [6,28] and lead to more sophisticated cyberattacks and databreaches [7]. Phishing kits, ready-to-deploy software packages sold at illicit markets for launching phishing attacks, have lowered the barrier of entry for malicious actors. Sellers often market phishing kits as bundles of quality-of-life features for attackers, such as built-in evasions from automated crawlers, exfiltration to Telegram channels, and obfuscation [45,60]. Deploying these kits can be as easy as uploading them to free hosting providers and mass sending multiple links that exfiltrate the credentials to an endpoint you control.

One of the selling points of phishing kits is evasion from researches and analysts, extending a phishing page's lifetime (delta time between deployment and discovery), and increasing the number of victims visiting the page without a browser warning. While server-side logic of kits can only make assumptions about the system based on the IP address and User agent, through API calls to the browser, client-side JavaScript code can query the user's system for CPU core counts, memory overhead, request user interaction, or call out to a 3rd party bot detection like CloudFlare[1] [64,66]. In the end, the JavaScript logic in a kit can range from a simple user-agent-based redirection to an AES-encrypted script that dynamically decrypts itself, identifies the browser through a series of API calls, and renders the page after confirming the victim is using a mobile device. Due to their mass-deployed and sold nature, phishing kits' exfiltration point, the targeted brand, or IP blocklists divide phishing kits into phishing kit families.

We leverage this complexity of the page's JavaScript logic to cluster the pages, revealing when they shared a phishing kit family, and further annotate the clusters with what client-side techniques they employ. Crawling phishing pages from APWG, PhishTank, OpenPhish, URLScan, and PhishingDB for *662* days using an instrumented Chromium-based browser, VisibleV8 [30]; we visit over 1.3 million pages and identify the 548 phishing kit families deployed behind 4,562 of these

[1]Similar to the phishing page that stole credentials from Troy Hunt, the maintainer of HaveIBeenPwned

pages. This paper aims to aid researchers and analysts by automatically differentiating groups of phishing pages based on the underlying kit, a task previously done through manual analysis. Researchers and analysts have different uses for this form of grouping, as researchers can accurately control for mass-deployed kits when studying evasions and other client-side techniques, and analysts frequently tie malicious actors to novel kits [8]. This work stands apart from prior research, as we automatically differentiate pages via all their browser APIs executed instead of static features or pre-trained classifiers. In doing so, we achieve a much higher accuracy than prior work, of F1=39.54% from [16]. Leveraging prior work into client-side cloaking and web privacy, we build a mapping of different evasion techniques (e.g., Fingerprinting, Obfuscation, CloudFlare Turnstile, and Client-side IP check) that a phishing page may employ and provide an up-to-date and mass-deployment-resilient description of how widespread these are. By clustering pages based on their browser API usage, we construct groups of pages based on a shared set of techniques. Evaluating these groups over a ground truth dataset of URLs to kit-family mappings, we find that clustering over the set of browser APIs executed yields an FMI-based accuracy of 0.97 and a validity score of 0.91. We then turn the clustering approach to 1,367,734 pages, which we group into *15,063* clusters. Figure 1 shows how the volume of pages/week can be by an order of magnitude when viewed as active clusters/week. We then use a predefined mapping of browser APIs to phishing techniques to better understand how widespread these practices are, while controlling for the mass deployment of a single kit. For example, Client-Side IP checks occur in 19,869 pages; however, this totals to 504 clusters, and 23% of the pages come from a single cluster, which consists of Facebook business account phishing pages. Overall, we offer the following contributions:

C1. We find that *browser API usage alone* is sufficient to isolate and distinguish known phishing kit families. With a ground truth dataset of 548 kit families deployed on 4,562 URLs, we achieve accuracy metrics of a 0.97% Fowlkes-Mallows score and an 0.91% V-measure in the clusters of these pages relative to their kits used.

C2. We experimentally show that browser APIs common on the web (DOM APIs, property reads, etc.) serve as a valuable identifier for identifying kits, as they signal the kit developer's choices, and that the more sophisticated a page's client-side logic is, the more indicative it is of the underlying kit.

C3. We isolate 744,903 phishing pages (out of 1.3M) with enough sophistication to sort into into *15,063* clusters, which allows us to observe that most clusters have short lives and target a single brand, suggesting that mass phishing campaigns gravitate towards low-quality, non-modular, and cheap or free kits.

C4. We highlight how widespread different techniques are in the ecosystem by annotating dynamic traces of pages within a cluster and propagating the behavior to the cluster as a whole. When annotating all the clusters for the different techniques used, we find that UI-interactivity and basic fingerprinting are near-universal. At the same time, mouse detection via browser APIs, Cloudflare Turnstile embedding, and dynamic script creation are still relatively rare. We also note that compared to prior work, we observe a decreased ratio of pages that employ pop-ups as a form of bot detection.

C5. We release a dataset of 4,562 phishing pages labeled with their underlying kit and an unlabeled dataset of browser APIs executed on over 1.3 million pages collected throughout *662* days.

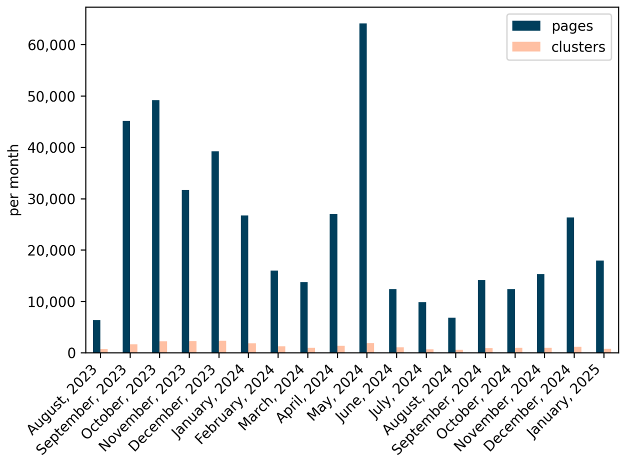


Figure 1: Comparison between monthly pages observed vs the monthly clusters observed based on dynamic behaviors. We see a drastic, non-linear reduction of phenomena that need to be investigated monthly.

2 Background

In this section, we provide a background on current developments in phishing as a phenomenon and adversarial JavaScript techniques, as well as an overview of the parties involved in the phishing ecosystem.

2.1 The Phenomenon of Phishing

Phishing is a form of social engineering where an adversary pretends to be a trusted entity to steal a user's credentials or gain access to a specific machine, network, or account to which the user has access. While delivery mechanisms vary, most phishing eventually leads to a webpage that requests some personal information (usually credentials) from the user. Because some legitimate websites use web fingerprints as a secondary authentication vector, phishers now also use browser fingerprinting APIs to identify real users and exfiltrate fingerprints to pair with stolen credentials [53].

Phishing is ever-evolving and still growing in prevalence. Groups that track phishing saw an increase in phishing domains in the 2020s. The most popular target sectors vary each year, but include software-as-a-service and webmail services (Q3 2020), financial services (Q3 2021), and social media (2024) [1]. In Q3 2024 alone, the Anti-Phishing Working Group (APWG) reported 900,000 phishing attacks.

As more enterprises and researchers study and combat phishing, phishers respond with new countermeasures to prevent automated crawling and phishing detection, collectively called "cloaking." If a phishing page determines the client is not a viable victim (e.g., a crawling bot, not in a specific country, etc.), it takes actions not to serve real phishing content. The page may halt with an empty DOM or redirect to a benign page, a long-dead phishing page, or an affiliate marketing page.

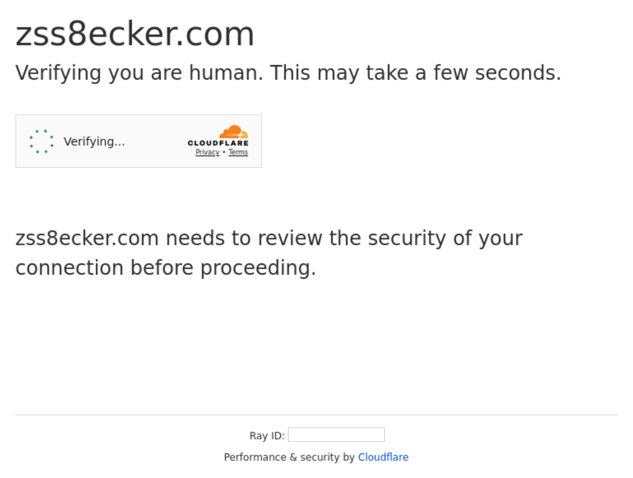
Cloaking techniques can be broadly classified into server-side and client-side techniques [45]. Server-side techniques are stealthier, but they rely on limited information about the client. Most server-side techniques rely on precompiled deny-lists or allow-lists of IP addresses, user agents, or referers (the page from which the link is visited as identified by an HTTP header).

Client-side techniques allow for richer evasion strategies but are also more detectable. Phishing pages use browser APIs to trigger permission pop-ups to identify crawling browsers, which often cannot interact with the whole browser UI. Because cloaking is technically very similar to legitimate bot-detection and abuse prevention, phishers use CAPTCHA and click-through pages as client-side cloaking. Recently, phishing pages have used Cloudflare Turnstile, a popular widget for abuse prevention, to identify automated browsers. Figure 2 shows a phishing page using a Cloudflare Turnstile when we crawled it.

Even the URL features of a phishing link contain techniques that have evolved to respond to anti-phishing research. Phishing pages frequently use URL shorteners (public or private) to obfuscate the final destination, landing pages requiring a user to follow hyperlinks to the actual page, and free web hosting with trustworthy top-level domains (TLDs).

2.2 Phishing as an ecosystem

Phishing is a logistical and technical challenge because a phisher must develop an effective phishing page with cloaking, find robust hosting for it, entice a victim to browse to it through SMS, email, or social media, exfiltrate the phished data, and then monetize the stolen credentials. This technical and logistical complexity, combined with interest from potential phishers, has created an underground economy to facilitate each step.


zss8ecker.com

Verifying you are human. This may take a few seconds.

zss8ecker.com needs to review the security of your connection before proceeding.

Ray ID:

Performance & security by Cloudflare

Figure 2: Example of a phishing page in our dataset that embedded Cloudflare Turnstile verification on a non-Cloudflare domain

Phishing facilitators sell bundles of customizable or ready-to-deploy phishing pages known as "phishing kits." They vary in features, sophistication, and cost. Phishing-as-a-service providers offer phishing kits and hosting services — essentially turnkey phishing systems. Both products lower barriers to entry. Prior work has shown that phishing kits may steal credentials from their customers' kit deployments [20, 39], adapt or "borrow" features from other kits [29], and they are sometimes tied to specific actors [10]. In this paper, when two kits share the same set of features, and only differ in minor additions to new IPs in the blocklists, or different directory structure, we refer to them as being the same "kit-family" Phishing systems may store credentials on the same server, risking loss when the page is inevitably taken down, but a more common practice is to send them to the phisher over instant messaging channels.

Credential sales markets simplify monetization. The credential sales part of the ecosystem has also adapted to modern MFA/2FA practices. With prior work showing that a browser fingerprint is enough to trick online services into triggering an MFA bypass [36], and has an increasing effect on the costs of stolen credentials [53].

2.3 JavaScript

Originally meant as a way of adding interactivity to webpages. The JavaScript ecosystem has evolved to allow varying low-level features to webpages through Browser APIs. HTML DOM APIs enable developers to modify page appearance, while LocalStorage and IndexDB allow write access to the browser's internal storage buckets; meanwhile, the File System API can allow access to the user's real machine's storage. Browser APIs can be function calls (or constructors), property reads, and property writes. Most of the privileged functionality comes from function calls.

The dynamic nature of JavaScript enables a variety of techniques for concealing itself from analysis and detection. JavaScript obfuscation can transform a known malicious sample into an undetectable one. Webpack enables bundling benign and malicious scripts and wrangling them to make static analysis harder. The other side of obfuscation is evasions; in addition to making code comprehension (via human or machine) harder, malicious actors have deployed time bombs, offloading parts of the malicious script to be read from the DOM or via a network request, and avoid detection. [49]

2.4 Hierarchical clustering

This paper utilizes hierarchical clustering, an unsupervised learning technique for segmenting data into nested structures (clusters) to identify pages that share phishing kits from their client-side behaviors. Specifically, we use HDBSCAN, a hierarchical variant of the density-based DBSCAN clustering technique. Advantages of HDBSCAN include not requiring prior knowledge of the number of clusters, no hyper-parameter tuning required out of the box[2]. HDBSCAN has been use by prior work for categorizing malware families [49, 57]. When we can access ground truth, we utilize the V-measure and Fowlkes-Mallows index (FMI) to validate our clusters. Both are scores of 0 and 1 that address how well the clusters map to ground truth classes. V-measure (Validity measure) is the harmonic mean between completeness (all members of the same class are clustered together) and homogeneity (clusters only contain members of a single class) [52]. FMI, on the other hand, is the geometric mean between precision and recall, providing a close analog for an F1 score in supervised learning. When we do not have ground truth for the pages, we use the silhouette score of all the clusters. Silhouette-score measures how well separated the clusters are, between -1 and 1. While usually used to fine-tune hyperparameters, we primarily use it to measure how well-formed the structures we extract out of HDBSCAN for all of the phishing pages are. Scores under zero signal overlapping structure, while scores above 0.5 and 0.7 indicate medium or firm separation, respectively.

3 Methodology

This paper provides a methodology for clustering using dynamic features, evaluating how closely the clusters resemble underlying phishing kits, and describing how widespread different adversarial techniques are in the ecosystem. The building blocks of our experimental design are browser API execution traces from phishing pages and a ground-truth dataset of pages where we know the underlying phishing kit. In the following section, we describe the experimental setup for gathering this data, the steps we took to aggregate and enrich the execution traces, annotate the clusters based on different techniques, and finally, the steps we took for clustering the data and evaluating them as analogs for phishing kits. An overview of our crawling and analysis pipeline can be seen in Figure 3.

3.1 Data Gathering

Our crawling infrastructure aims to ingest phishing URLs from upstream providers and output execution traces from the page, as well as a potential kit used for that page.

URL feeds: We gathered phishing pages by monitoring the following phishing feeds: OpenPhish [48], PhishTank [19], URLScan [55], SMS Gateways [43], PhishDB [41], and APWG [13], based on the availability of the feed and the level of access we had at the time. Every two hours, we checked these feeds for new URLs (limited to the last 48 hours) and submitted them into two different crawlers: VisibleV8 and *KitPhishr*.

VisibleV8: To get execution traces for every script loaded when visiting the page, we used an automated Chromium-based crawler with VisibleV8 patches applied [30]. The VisibleV8 patches modify Chromium to output a log of all JavaScript APIs executed for every script loaded on the page. We automate the browser to visit the page and take screenshots with puppeteer [26], an NPM package by Google to help in UI/UX testing and browser automation. The patched Chromium crawler uses puppeteer-stealth, a set of configurations to help mask the headless Chrome and Puppeteer itself from detection tools [47]. We initiated the crawls from a network designated for research purposes, for which the ISP would register as 'educational' for any IP intelligence API. We used Catapult [25], a man-in-the-middle proxy, to capture the entire HTTP archive for replayability. The crawler stays on the page for 45 seconds before taking a screenshot to allow scripts to load and start executing, consistent with prior work [22, 30].

KitPhishr: While not guaranteed, some malicious actors leave the zip files of the kits used in a discoverable folder on the same server that hosts the website (for example, the Apache document root). *KitPhishr* [2] is a Go-based URL fuzzer that attempts to identify any leftover zip files on the server. Prior work [37, 45] establishes *KitPhishr* as a method for collecting and analyzing phishing kits. If successful, it will download the zip file and make a note of the domain from which *KitPhishr* acquired it.

3.2 De-duplication

Once we have collected browser API traces and potential phishing kits, we post-process the traces into a set of APIs executed in a 1st-party context per page and de-duplicate the phishing kits based on archive file similarity.

[2]Unlike DBSCAN, where you have to optimize towards an ϵ HDBSCAN picks the most stable clusters based on a center of mass

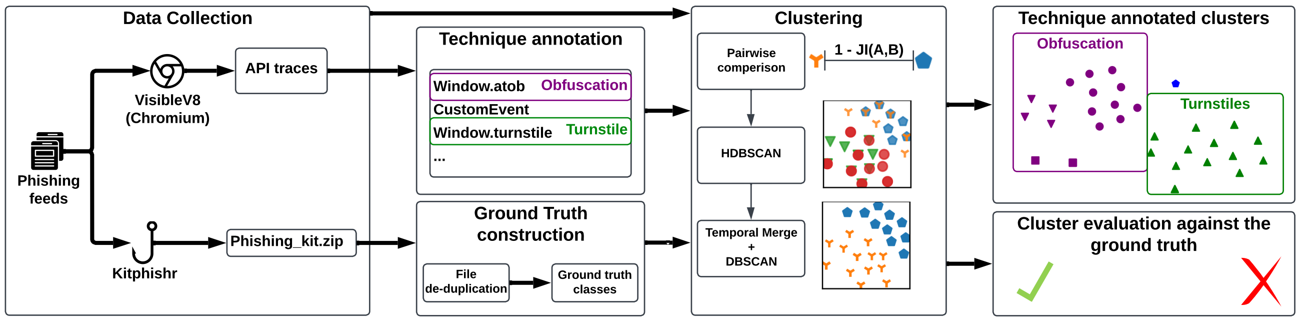


Figure 3: Crawling and clustering infrastructure

3.2.1 Trace postprocessing

VisibleV8 has a default log-postprocessor set. These programs take the raw lags generated by the patched Chromium browser and convert it into an organized database, identifying duplicate scripts via SHA3 hash, clearly marking the origin of each script that loaded and isolating which JavaScript API calls are browser API calls defined in the WebIDL file[3] generated while building the patched Chromium. For our analysis, we extracted tuples of the original page's URL, the script's URL, and an unordered set of APIs executed by this script.

To not introduce artifacts into our clusters from cloaked pages and 3rd-party scripts like Google Analytics, known to be present in phishing pages [35], we isolate API sets executed by 1st-party scripts (from now on called 1st-party API sets). We establish the root domain as the domain submitted to the feeds and the origin as the domain from which the script is loaded. We consider a script 1st-party only if it is hosted on the same domain we acquired from our feeds (root domain). The only exception is when we identify which pages embed a Cloudflare Turnstile script. As some of the Turnstile scripts are hosted on 'Cloudflare.com.'

Identifying Kit-families: We crawl the URLs with *KitPhishr* to establish a ground truth dataset with URLs originating from the same kit[4]. For zip files extracted via *KitPhishr* that have password-based encryption enabled, we use the SHA256 hash of the zip files to identify which domains yielded the same kit. For the remaining zip files, we further de-duplicate them into Kit-Families, examining them as a set of SHA256 hashes of source files[5]. If the Jaccard index-based similarity, where $JI(A,B) = |A \cap B|/|A \cup B|$, of these sets is equal to or greater than 80%, we consider the two kits to be from the same family, and thus group all domains from both kits into the same group. For any similarity between 10%-80%, 5 reviewers manually inspect the kits, looking for similar file contents and directory structures, to classify if the two are in the same kit-family.

[3]https://developer.mozilla.org/en-US/docs/Glossary/WebIDL

[4]For any domain that yielded two or more zipfiles, we discard all

[5]Source files identified via a python Magika module [24]

Adjusting for Cloudflare: Many anomalies (multi-layer evals, non-deterministic behavior, dynamically updated scripts) originate from Cloudflare scripts on the same domain as the phishing pages. Since Cloudflare scripts load from a URL with a *'cdn-cgi'* in the path, for most of our analysis, we do not consider scripts loaded from that endpoint.

3.3 Identifying kits

We hypothesize that similarity in browser API execution means that the pages originate from the same phishing kit. To test this, we ingest API sets from pages for which we were able to identify phishing kits and output a potential clustering of those pages, checking how well the clustering maps back to the ground truth information.

To establish this similarity, we use the Jaccard index on the API sets that 1st-party scripts execute. We use Hierarchical Density-Based Spatial Clustering of Applications with Noise [15] (HDBSCAN), with a minimum cluster size of 2, to cluster the pages with the Jaccard distance as our distance kernel. With HDBSCAN requiring minimal fine-tuning out of the box, and requiring no prior knowledge of the number of clusters we need to look for, we use the ground truth labels from *KitPhishr* to evaluate the clustering. When ground truth is available, we evaluate the clustering using the *Fowlkes-Mallows Index* [23] and *V-measure*. We use sklearn's measure module to calculate all cluster evaluation metrics [50], and separate all the noise elements into singleton clusters for evaluation, as removing all noise samples would give it an unreasonable advantage; however, keeping the elements in the same noise cluster would artificially reduce the homogeneity score. For the unlabeled set of pages for which we do not have kit labels, we use silhouette score[6], a metric for how well packed and separated the clusters are, to evaluate the clustering when we do not have ground truth.

[6]While silhouette score is biased against non-convex clusters, based on our results, we do not see it necessary to switch to a density-specific cluster metric

To process 1,367,734 pages would require a distance matrix of distances (64-bit float) over a 1.5Tb in size. To resolve this restriction, we first divide our data using a 4-week rolling window, rolled by 2 weeks, and cluster pages within that window. We refer to these clusters as ‘local clusters’. However, these local clusters can not represent phenomena like the re-emergence of kits if it happens after 2 weeks and contains duplicates of the same page. To resolve this, we merge any two clusters with at least 1 page in common between local clusters and merge them using the representative API set for the clusters (API set intersection of every page in the cluster). We use DBSCAN with a conservative $\epsilon = 0.03$ to merge them. Intuitively, this ϵ represents the maximum dissimilarity tolerated between clusters.

3.4 Data enrichment

In addition to all the data gathered, our analysis references a manually crafted Browser API to phishing technique mapping, JavaScript execution traces for the top 5 brands' login pages targeted in our dataset, and characterization of cluster lifetimes and deployment diversity.

Technique to Browser API mapping: Client-side JavaScript can engage in data harvesting (exfiltrating information dynamically and not through form submission), evasion (conditional dynamic behavior aimed at hiding functionality or contents), obfuscation (unconditional behavior meant to hinder static analysis), and mimicking (dynamic behavior to make the page more believable, for example, false loading pages, stage by stage data extraction). Based on prior work by Su *et al.* and Zhang *et al.* and manually identifying APIs from the Mozilla Developers Network (MDN) documentation, we present a table mapping standard phishing techniques to browser APIs in Table 6.We leverage the presence of these APIs in the execution traces as a signal of the technique being present in the page. If the page falls within a specific cluster, we mark the entire cluster as using that technique. We use this to combat the phishing pages that engage in non-deterministic behavior. For Cloudflare turnstiles embedding, we use a non-browser API as our detection metric, as Cloudflare's native turnstile script will read the value of `Window.Turnstiles`. For Client-side IP checks, we manually looked through every `Window.fetch` and `XMLHttpRequest.open` argument URL given that argument was present in more than 50 phishing pages, and identified 15 that were IP reputation APIs.

Brand's Original page: OpenPhish and APWG's eCrime Exchange report the brand that a phishing page targets. We selected 5 out of the top 52 brands targeted based on popularity by page number and brands that represented seasonality targeted sectors (like the IRS or banks). We collected VisibleV8 logs for their home pages and, when applicable, their login pages, to assess how similar phishing pages are to their target pages.

Cluster lifetime and deployment diversity: Throughout this work, we refer to cluster lifetime as the time range between when we observe the first page belonging to the cluster on a phishing feed and when we observe the last page belonging to the cluster. We measure deployment diversity of the phishing pages by looking at the effective 2nd-level domain (e2LD) for the URLs. Using the e2LD instead of the entire hostname ensures that pages deployed on ’pages.dev‘ or ’blogger.com‘ are considered a single deployment form.

4 Data characterization

This paper utilizes two distinct datasets. Labeled dataset of phishing pages deployed in the wild, their corresponding phishing kit, and an unlabeled dataset of phishing pages.

4.1 Labeled dataset

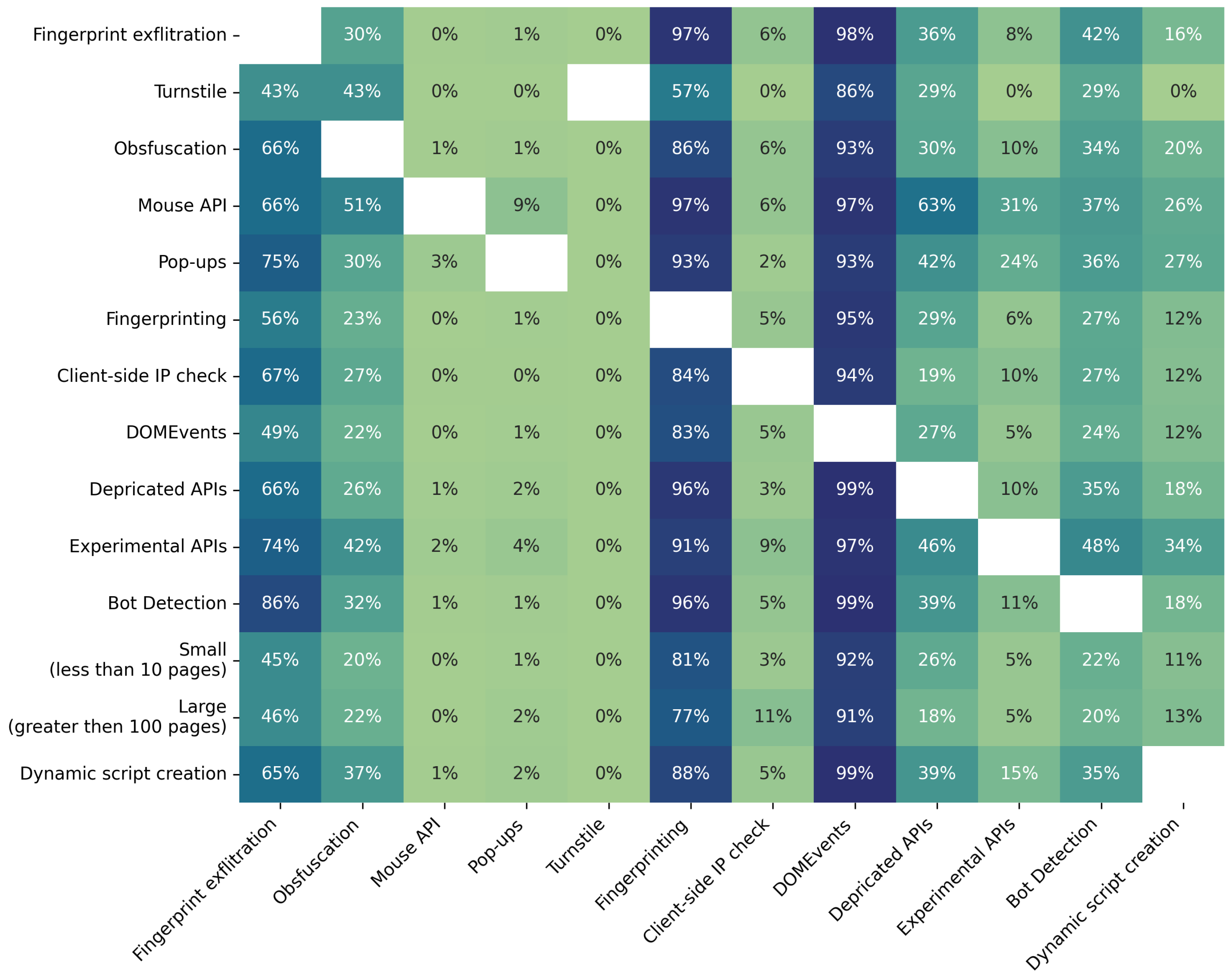
The ground truth data for this paper is a mapping of 548 kit families to 4,562 phishing pages. We collected 7,273 archive files by running all 1,367,734 through *KitPhishr*. Only two were encrypted, which we treated as unique kits based on their SHA256 hash. We further filtered the archive files by ensuring they contained at least a single code file, as Magika [24] identified. This reduces our archive count to 2,262. Finally, we group these kits into 548 families, 5,871 pairs of kits having to be manually inspected. The most deployed kit was “2e94aff28a2c” (1,073 URLs), which made use of 3 seperate server-side blocklists (.httaccess file and two seperate PHP modules with regex rules for user-agent and IPs) and called 52 distinct browser APIs. The kit obtained throughout 545 days of the crawling (from 256 distinct urls) was “fce61e98018d”, a USPS phishing kit which executed on average 170 distinct APIs, with server-side and client-side IP checks, advance fingerprinting API calls, obfuscation, and javascript generated DOM (Vite).

In line with prior work, 89% of these phishing kits were written in PHP, and we found 11 contained Python code. During our manual analysis of these kits, we concurred with prior work that many of these kits reused each other's code, especially regarding server-side IP blocklists. We found two module-like anti-bot detection files frequently redistributed across kits.

Finding 1: *Phishing pages from different kit families have vastly different browser API usage*. Pages from different kits have an average API similarity of 15.9% ($\sigma = 0.2$). On the other hand, we find that the pages from the same kit family on average have an API similarity of 98.6% ($\sigma = 0.1$).

4.2 Unlabeled Phishing pages

In total, we crawled for *662* days, collecting browser traces from 1,816,827 urls (1,392,821 domains). These included false Facebook suspension, USPS mail delivery, tech support



	Fingerprint exflitration	Obsfuscation	Mouse API	Pop-ups	Turnstile	Fingerprinting	Client-side IP check	DOMEvents	Depricated APIs	Experimental APIs	Bot Detection	Dynamic script creation
Fingerprint exflitration		30%	0%	1%	0%	97%	6%	98%	36%	8%	42%	16%
Turnstile	43%	43%	0%	0%		57%	0%	86%	29%	0%	29%	0%
Obsfuscation	66%		1%	1%	0%	86%	6%	93%	30%	10%	34%	20%
Mouse API	66%	51%		9%	0%	97%	6%	97%	63%	31%	37%	26%
Pop-ups	75%	30%	3%		0%	93%	2%	93%	42%	24%	36%	27%
Fingerprinting	56%	23%	0%	1%	0%		5%	95%	29%	6%	27%	12%
Client-side IP check	67%	27%	0%	0%	0%	84%		94%	19%	10%	27%	12%
DOMEvents	49%	22%	0%	1%	0%	83%	5%		27%	5%	24%	12%
Depricated APIs	66%	26%	1%	2%	0%	96%	3%	99%		10%	35%	18%
Experimental APIs	74%	42%	2%	4%	0%	91%	9%	97%	46%		48%	34%
Bot Detection	86%	32%	1%	1%	0%	96%	5%	99%	39%	11%		18%
Small (less than 10 pages)	45%	20%	0%	1%	0%	81%	3%	92%	26%	5%	22%	11%
Large (greater then 100 pages)	46%	22%	0%	2%	0%	77%	11%	91%	18%	5%	20%	13%
Dynamic script creation	65%	37%	1%	2%	0%	88%	5%	99%	39%	15%	35%	

Figure 4: Confusion matrix between all of the techniques enumerated and cluster lifetime characteristics, normalized by row.

scam, and AT&T login pages, and more. All the pages collected have been labeled as phishing, as they pose as a trustworthy entity to gain credentials or network access from their victims, despite their varying tactics and credentials collected [5]. Only 1,367,734 qualified for clustering by executing at least 8 JavaScript APIs in a first-party context. 451,634 pages did not execute any APIs in the first-party context, and only 744,903 executed at least 8 APIs to qualify for clustering. We first processed the pages into 36,708 local clusters. Figure 6 shows the distribution of the silhouette score of these local clusters. With an average score of 0.8, these clusters are well-formed. Once clustered together, the average cluster in our dataset contained 49 pages ($\sigma = 483$), existed for 101 days ($\sigma = 169$), and executed 56 ($\sigma = 54$) APIs.

We remove 70,142 pages out of the clustering, as they formed "malformed clusters", clusters with fewer then 2-APIs in common between all pages, this was done to avoid miss-characterization of different technique across the ecosystem. In addition to these, Cluster-"31885032", by far the largest by page count is excluded from our evaluation, as it mainly comprised of CloudFlare notices stating "sorry, you have been blocked", mixed with pages that dynamically generate their frontend.

After manually inspecting a sample of clusters, we observed that these clusters have unique pages across deployment types (AWS, Cloudflare, DigitalOcean, etc.) and languages. For example, Cluster-53d5c420 comprises 483 pages across five unique e2TLDs and contains pages engaging in voice-based phishing attacks (tech support scams), varying phone numbers and error messages in each, shown in Figure 10. With over 57 APIs in common, it is clear that these pages' usage of keyboard intercepting APIs, Audio APIs, and Network APIs calls (to ipwho.is) for IP intelligence caused the pages to cluster together.

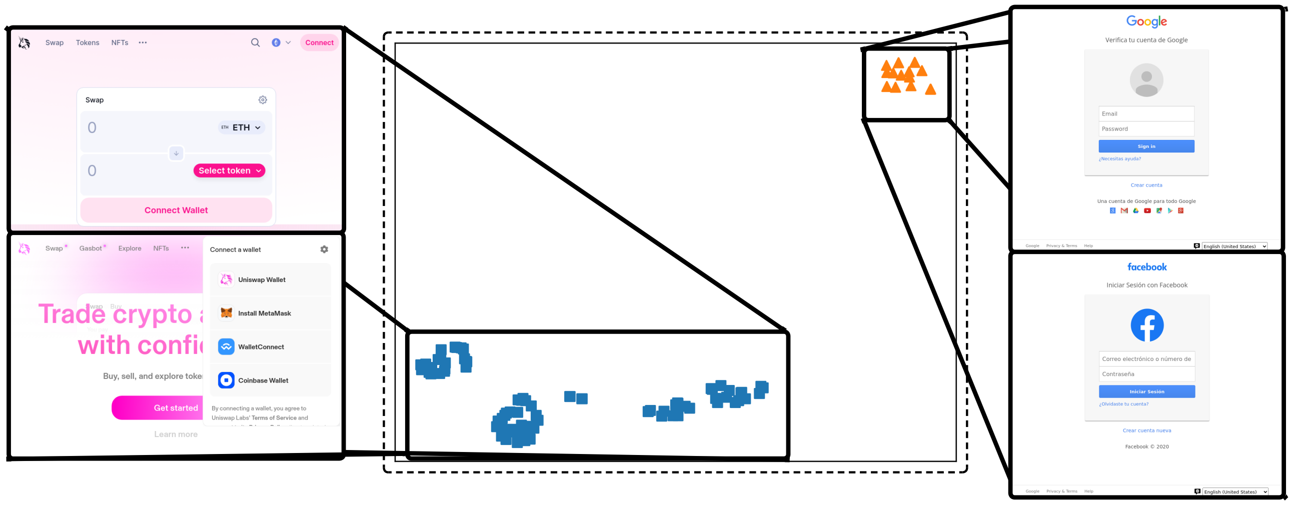


Figure 5: Example of pages from two different pages and embedding between them. We used t-distributed stochastic neighbor embedding to visualize the distance matrix between all pages in a 2-D plane. The clusters presented here are from Ethereum wallet pages with a few variations of the landing page (▲) and pages that descend from an open-source phishing kit discussed in Section 5 (🟦). We should note that while the pages have different logos on the right-hand side, the bottom left corner still reads "Google" on those pages.

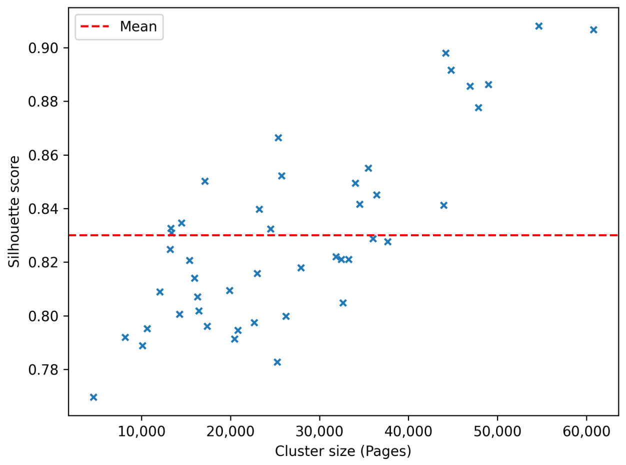


Figure 6: Distribution of the Silhouette Score of local clusters vs. their size (sum of pages)

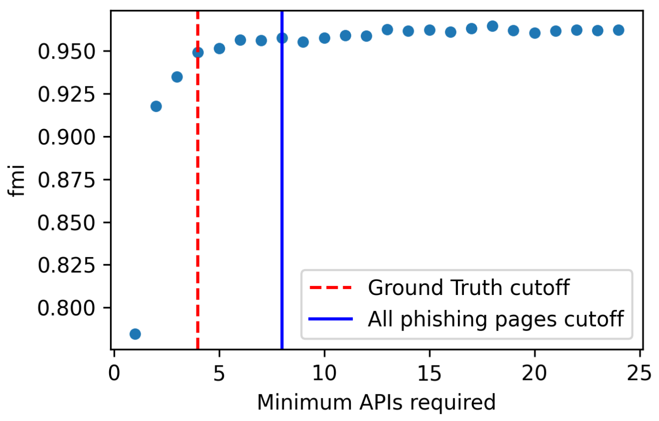
Figure 5 shows two different clusters where pages have different DOM elements or landing pages being grouped, with example screenshots pulled from both. One cluster is from a deployment we can tie to a public GitHub repository (discussed in Finding-11), and the other is a collection of crypto-wallet targeting pages. The figure shows a t-distributed stochastic neighbor (t-SDN) embedding for the distance matrix between all the pages from the two clusters to illustrate better how the pages are separated.

5 Results

5.1 Kit identification

Finding 2: *In majority of the cases, pages that execute at least two distinct browser APIs can be related to one another based on the underlying phishing kit; as the sophistication of the page grows, so does the accuracy of the clustering*. We find that clustering all pages that execute at least two browser API, yields an FMI based accuracy of 0.92. Figure 7 shows the V-Measure and FMI for our clusters as we increase the requirement of distinct APIs in the execution trace. We ultimately chose four browser APIs as the requirement for further experimentation on the ground truth data, as they provided a good tradeoff between V-measure and the number of pages used. For the unlabeled pages, we used 8 APIs, as they provided zero malformed clusters on our ground truth data (clusters with no API sets in common). Clustering pages from 4,562 pages across 548 kits, yields 654 clusters. Evaluating these clusters against the ground truth labels for each page, we find that our clusters have an FMI-based accuracy of 0.97 and a V-measure of 0.91 (with completeness and homogeneity scores of 0.90 and 0.92 respectively). Pages with ground truth labels for originating kits remain appropriately sorted in the final clusters. We maintain an FMI of 0.95 and a V-measure of 0.89, respectively.

Finding 3: *Browser API sets better separate phishing kits than script hashes, even when we include scripts dynamically extracted from eval statements*. There is no prior work in



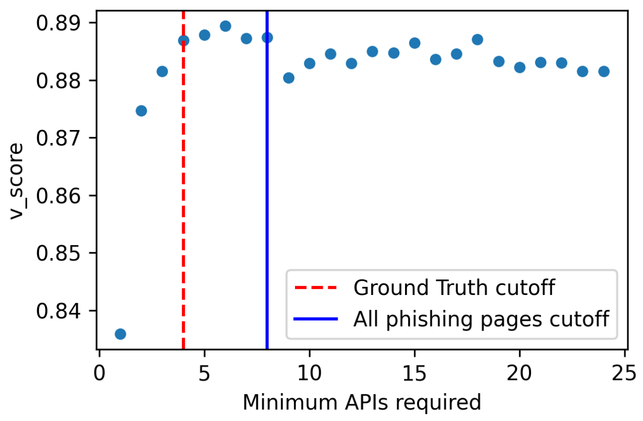


Figure 7: Validity measure for clusters vs. minmum distinct APIs required for clustering

Table 1: Comparison of the evaluation metric when Script hashes are used instead of APIs executed

Method	FMI	V-Score
Dynamic	0.95	0.89
Scripts (No eval)	0.88	0.85
Scripts (No eval, 1st party)	0.80	0.81
Scripts (1st party)	0.81	0.81
All script	0.89	0.85

automatically distinguishing pages based on their underlying kit, without prior knowledge of the kit's file structure, so we use sha256 hashes of executed scripts as features into HDBSCAN to compare against our approach. Table 1 shows that browser API sets maintain a higher accuracy than SHA256 hashes of scripts, even for scripts that are extracted out of eval statements, and would require dynamic or static analysis to acquire.

Finding 4: *DOM APIs and property reads are a valuable signal in kit differentiation*. We find that removing DOM-related APIs or property reads out of consideration drastically reduces the number of pages we can consider for ground truth evaluation without increasing our overall accuracy. The key insight here is that these APIs signal particular choices the kit author made about the UI library they used, if they chose to hide the DOM as an evasion, or not draw it to begin, or which selectors, IDs, or classes they prefer to use. Clustering evaluation metrics for the ground truth dataset with DOM, SVG, and CSS APIs removed results in an FMI of 0.93 and a V-measure of 0.86. When all property reads were removed, we saw an FMI and V-measure of 0.93 and 0.85, respectively. In both cases, we can cluster fewer pages and thus identify fewer kits, but we do not see a significant improvement in clustering accuracy.

5.2 Clusters in the wild

Finding 5: *69% of clusters contain URLs only marked by a single target brand by our threat intel sources*[7]. This phenomenon is observed in the ground truth dataset, as URLs for 90% of the kits were targeting a singular brand. Together, this provides strong circumstantial evidence that deployed kits are increasingly becoming brand-specific. 1,253 clusters (19%) of the clusters had two brand labels. However, the most popular combination of these was "Meta/Facebook", "Facebook/Instagram", "National Police Agency JAPAN/Facebook", and "holiganbet/jojobet", keeping the parent organization of the target the same in the majority of the cases. Manual examination of clusters with "National Police Agency JAPAN/Facebook" brand labels revealed shopping pages in Japanese to be mismarked with that label from our data feeds. The cluster with the most diverse set of brand labels had 14 unique brand labels, which was a cluster with 12,467 pages with simple sign-in pages that exfiltrated information using client-side registered event listeners. Furthermore, we find that 16,100 pages (3% of the pages observed spanning 313 clusters) come from phishing kits collected using KitPhisher; however, we could not pull the kit from the URL in all pages. Figure 8 shows a histogram of the unique brand labels observed per cluster.
Finding 6: *Top 50 clusters by page count, account for over 40% of the phishing traffic observed. 40.7%* pages are sorted into one of the 50 top clusters. The 10 clusters by page count are 20.8% of the total pages alone. We provide a breakdown of these clusters along with manual labeling of what campaigns they correspond to in Figure 9. We note that one of the clusters, which we labeled dynamically generated, turned out to be a noisy cluster of simple pages that dynamically generate the page using JavaScript, without any sophisticated client-side behavior, based on their browser API traces. The E-commerce cluster shown in Figure 9 has a significant seasonality, with

[7]We did not include clusters in this count that had no brand-labeled URLs in them

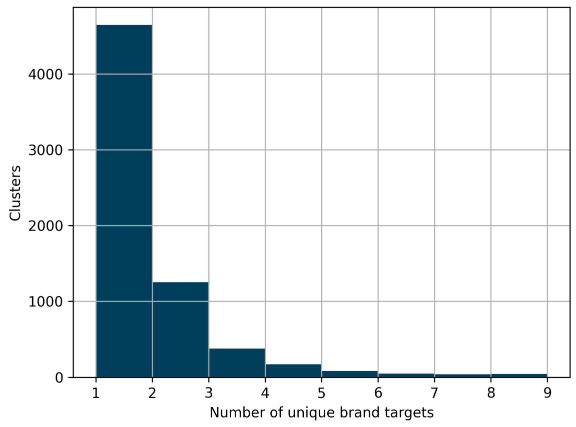


Figure 8: Distribution of unique brand labels per cluster (only counting clusters that had at least one page with a brand label)

an auto-correlation function showing significance at lags 7,14, and 21; meaning the appearance of the clusters on the feeds happens every week. However, further investigation showed that the majority of seasonal clusters are similar e-commerce phishing clusters, with vastly different dynamic behavior, allowing us to conclude that the seasonality in the majority of the clusters is due to regular reporting by threat-intelligence sources to our feeds, and not seasonal deployments of kits.

Finding 7: *72% of the clusters (144,695 pages), are only seen for a single month by phishing feeds*. Lack of re-emergence for signals either the deployment of throw-away kits, or the possibility that these clusters engage in profiling of phishing feed detectors as described by [12]. 15% of the clusters (14,555 pages) we observe are only seen for one day, along with brandless bank pages and a phishing page impersonating the government of Korea. Meanwhile, 1,282 clusters (11%) have lifetimes longer than 100 days. Some clusters with a lifetime greater than 100 days still only deploy a few pages (less than 1 page every 10 days). We use this as a heuristic to identify 251 clusters 're-emergence' through our observation period.

5.3 Phishing Techniques across clusters

Finding 8: *UI interactivity and fingerprinting are a near-universal behavior across clusters*. Multi-stage phishing pages are very well documented in prior work, and we find that most clusters (91%) register a click event listener using JavaScript. Though this could be as simple as submitting credentials using JavaScript, this highlights the need for researchers to augment their crawlers in the future to extract better and more complete execution traces from websites. We split fingerprinting into two categories, basic and advanced. Basic fingerprinting, which follows the list of APIs identified by Zhang *et al.* in [64] was present in 80% of the clusters (over 300,000 pages), and Advance fingerprinting (measured by at least 5 APIs idenfied by Su *et al.* in [58]) show up in 70% of the clusters. Together, 85% of clusters (9,572 clusters, 313,212 pages) exhibit some form of fingerprinting.

Table 2: Breakdown of the obfuscation techniques observed in our dataset

Obfuscation techniques	Pages	Clusters
Window.atob	61,125	1,455
eval	14,561	982
Textdecoder.decode	11,113	534
SubtleCrypto.decrypt	1,185	36

Finding 9: *Fingerprint exfiltration, obfuscation, and bot detection are widespread across clusters*. While fingerprinting is near universal, we find that a smaller fraction of the clusters employ obfuscation, fingerprint exfiltration, and timing for bot detection. Prior work has shown interest in these behaviors [36, 49, 53, 65], meaning kits that forgo this may be rudimentary either by negligence or design, to avoid static and dynamic phishing detection based on JavaScript features. Dropping anti-bot detection features has been observed before, with an Office365 phishing kit (dubbed Tycoon2FA), opting to remove CloudFlare Turnstile integration, as it was being used as a feature for detection [8].

Another common tactic for bot detection is timing-based checks; this can be done via Browser APIs by calling `Performance.now` right before and inside of a `Window.setTimeout` statement to measure the time differential between setting the timeout and its triggering. We find that 22% of clusters call `Performance.now` in conjunction with `setTimeout`.

On the other hand, 31% of clusters (2,395) employ some form of obfuscation. We present the breakdown of all obfuscation techniques in Table 2 and as we can see, eval and Base64 encoding were the most popular ways of obfuscation. Despite the best recommendations to web developers to avoid using 'eval' [9], JavaScript's eval function remains a favorite for obfuscation and evasions [49]. Sometimes, a script is executed via 'eval()', which evaluates yet another script itself; we measure this phenomenon as a level in *eval-depth*. We find that 48 clusters have pages that go to eval-depth 3. However, this seems to be a side-effect of embedding the phishing pages (mainly ones targeting Facebook) in Blogger.com pages.

Finding 10: *While rare, client-side IP reputation checks are present across multiple clusters*. While only present in 504 clusters (19,869 pages), we identify 15 unique IP reputation APIs used by phishing pages as soon as the page loads. We present a full breakdown in Table 3. While not the most popular, *api.ipregistry.co* presents an interesting case study, as it enables the identification of educational networks. Manual examination of pages from these clusters reveals snippets sim-

```
await this.$http({
  method: "get",
  url: "https://api.ipregistry.co/?key=" +
   ↪  this.key
}).then(e => {
  const s = e.connection.type,
      c = ["cdn", "hosting", "education"];
    /*omitted for brevity*/
    if (c.indexOf(s) != -1)
        /*Redirect*/
})
```

Listing 1: Example of IP-based cloaking by using a 3rd-party reputation API. The following example specifically cloaks away from educational networks like ours.

Table 3: List of all API endpoints that client-side code reaches out for IP intelligence.

API url	clusters	pages
api.db-ip.com	40	3,124
api.geoapify.com	3	63
api.ipapi.com	5	143
api.ipgeolocation.io	21	96
api.ipify.org	177	7,417
api.ipregistry.co	9	3,995
freeipapi.com	38	6,444
geolocation-db.com	14	492
geolocation.onetrust.com	37	364
get.geojs.io	14	753
ipapi.co	106	735
ipinfo.io	65	1,963
ipwho.is	47	798
pro.ip-api.com	20	101

ilar to Listing 1, which conditionally chooses to redirect away from cloud hosting providers, content delivery networks, and educational networks, like the one we performed the crawls through. However, due to them employing other browser APIs, before the cloaking behavior, we are still able to cluster the pages based on the initial logic of the landing page.

Finding 11: *Pop-UP APIs are declining in usage*. We see only 104 clusters (1,323 pages) call out to pop-up requesting APIs. Among these, Geolocation.getCurrentPosition (55 clusters) was the most popular. While requiring a pop-up to interact with, this API can also be crucial in cloaking, as any VPN or proxy does not mask the results.

We observe a smaller fraction of the ecosystem (16 clusters, 148 pages) than [64] employs this cloaking technique, especially when it comes to triggering a notification pop-up to verify user interaction. This could be a result of Firefox, citing low engagement with the notifications, starting to require user interaction to trigger the popup [42] at the end of November 2019, when crawlphish's data collection ended. Chrome has since discussed modifying the notification API to make the request less disruptive to the user experience [3]. The Lack of pop-up requests could also be explained by our *Finding 9* regarding the usage of IP intelligence APIs or by the overwhelming amount of pages (67%) registering at least one HTMLElement event handler, which would be classified as a *Click-through* by Crawlphish's taxonomy. We report a full breakdown of APIs related to the Crawlphish categorization of client-side cloaking in Table 5.

Finding 12: *Mouse Detection API calls and Cloudflare Turnstile embedding are specific to a small group of clusters*. While supported by most modern browsers, we see a very rare use of Mouse Detection APIs. Only 35 clusters employ mouse detection-related APIs. Two of these clusters[8] are from an open-source phishing kit, leveraging botguard, and are from a public GitHub repository, which was last updated in 2017[9]. We see these clusters deploy across 17 unique domains, starting from 2023-10-07 all the way to 2024-07-19. *7 clusters (181 pages)* embed a Cloudflare turnstile check in their page; some domains are not hosted on Cloudflare. It should be noted that in the case of redirection. At the same time, we discussed the presence of WebAssembly-based captchas for bot detection. Embedding a Cloudflare Turnstile check allows the phishing kit authors to offload bot detection to a well-established ecosystem. Recently, analyses have identified high-value phishing kits with this behavior [4]. However, subsequent analysis of the same threat actor identified a shift from turnstiles to HTML-Canvas drawn captchas.

Finding 13: *The phishing ecosystem consists of clusters that utilize both cutting-edge, experimental browser APIs, and extremely deprecated APIs*. We find 421 clusters (8,270 pages) that utilize *NavigatorUAData.getHighEntropyValues* for fingerprinting and *Keyboard.lock* to restrict user input, APIs not fully supported by Firefox and Safari. We identified 10 clusters across 4,433 pages that used *Scheduling.isInputPending*, however, upon closer inspection, these were not pages using a novel kit, but instead pages that used Google Sheets to construct their landing page. On the other hand, 25% of the clusters spanning (47,001) pages use a deprecated API.

While not experimental, WebAssembly is still a relatively modern web practice. We find 199 clusters (5,107 pages) that use WebAssembly related APIs. Upon manual inspection of 33 unique WASM modules on these pages, we identify bot-detection, in most cases, by using FriendlyCaptcha, as the most common use case for WebAssembly in phishing.

Figure 4 shows the confusion matrix between the techniques we enumerated and the size of the clusters. We see no noticeable difference in techniques regarding cluster size, except for a higher percentage of large clusters using client-side IP checks. We also observe that pages that employ experimental

[8]Split due to infrastructure inability causing crashes in early crawls
[9]https://github.com/ashanahw/Gmail_Phishing

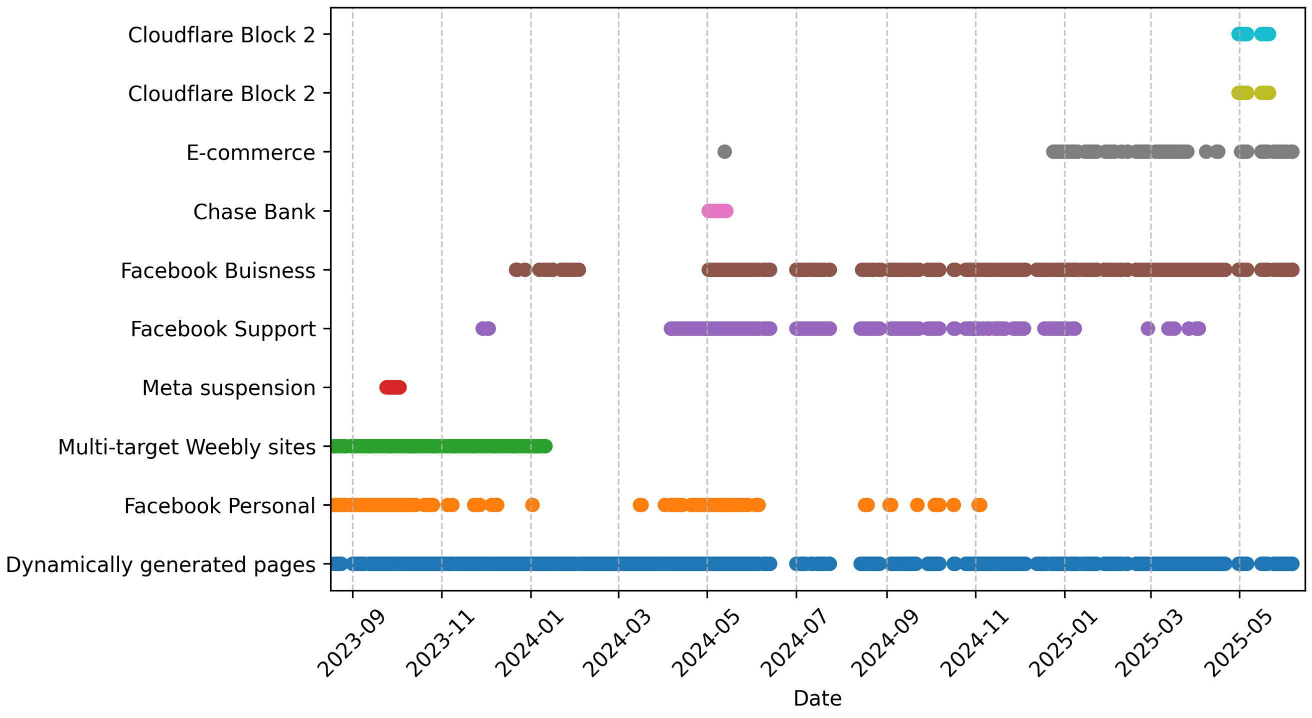


Figure 9: Timeline of the top 10 clusters based on the number of pages

APIs also tend to include a deprecated API call, which aligns with their usage for browser fingerprinting, rather than novel cloaking logic.

6 Discussion

6.1 What makes a kit identifiable?

Sets of browser APIs used are not the most granular method to describe pages; however, the more sophisticated the phishing kit becomes, the easier it is to spot by just the browser APIs it uses, as everything from how it chooses to evade researchers to how the UI libraries they use build the DOM, can make it stand out. In plain JavaScript, without browser APIs, a page can not reach a C2 server, gather browser fingerprints, dynamically generate or cloak page contents, request browser pop-ups, or even respond to a button click. In the case of the pages clustered in Figure 10, the APIs *Keyboard.lock*, *HTMLDocument.onkeydown* for keyboard locking, *Window.atob* for obfuscation, and a handful of fingerprint APIs and DOM APIs for dynamic content generation, set these pages apart from the other clusters.

Finding 14: *Phishing pages vary wildly from the brand that they are mimicking*. We collect browser API traces from Facebook, USPS, Meta, Microsoft, and IRS login pages and compare them to their phishing counterparts. The average similarity of the APIs executed by the phishing page and the original page is 11%, indicating that browser APIs do not relate to the target page. We found no pages where the original page's API set was a subset of the phishing page's API set, and in 5% of the cases, the phishing page executed at least half of the APIs from the original page. We report per-brand findings in Table 4 and note that the least similar brand was USPS, which could be the result of USPS phishing pages being multi-stage pages requiring user interaction and targeting credit card information [17].

This, however, does not indicate that browser API usage alone is a good indicator of maliciousness. All of the techniques described in Section 3 are common throughout the web. Phishing pages are singular in their purpose, and their choice to engage in these techniques sets them apart.

6.2 Why cluster phishing pages?

As shown in the examples in this paper, phishing feeds can be a noisy data source for studying the overall ecosystem. From e-commerce pages to mass-spammed USPS and EZ-parking phishing pages, differentiating between a handful of pages, the methodology in this paper is aimed at researchers and analysts. For research, identifying kits in a dataset of phishing kits helps control for easily obtainable or mass-deployed phishing kits, measuring the prevalence of different techniques across kits, rather than pages. Features extracted from dynamic execution with unsupervised learning for establishing when two

Table 4: Comparison of browser APIs executed by phishing pages against the original login page of the brand they are targetting

Brand	Phishing pages observed	Avg % Similar	Std Dev (%)	Perfect Subset	50%+ Match
Facebook	338,686	11.6	9.6	0	24,039
USPS	66,692	10.3	10.5	0	53
Meta	51,969	12.4	7	0	2,605
Microsoft	3,768	11.8	8.7	0	419
IRS	1,207	11.5	9	0	61

pages are similar allow for resilience towards run-of-the-mill obfuscation of the client-side code.

For analysts, our methodology acts as a quick way to aggregate and share phishing kits-related threat intelligence between pages. Things like server-side cloaking technique, preferred exfiltration method, ties to APTs, and data exfiltrated. While kit-families can vary in which IPs they denylist, and what user-agents they allow, fingerprinting the underlying kit can allow analysts to deduce if a page employs these techniques in the first place.

7 Related Work

7.1 Phishing

Phishing detection: There is a wealth of research on phishing detection as the tug-of-war between adversaries and security professionals continues. Recently, Liu *et al.* and Abdelnabi *et al.* have deployed vision-based techniques to detect phishing pages [11,37,38]. [38] also presents over 6,000 phishing kits analyzed as part of the work. With adversarial attacks ensuring that the page looks different to crawlers and analysis, some have turned to extracting features from the URLs themselves, more recently via LLMs in [18,31] and earlier via statistical models and machine learning in [33,51,56,63]

Studying and combating adversarial techniques: Divakaran *et al.* in [21] reaffirms the need to keep up with the latest adversarial techniques to build better detection systems for phishing. Prominent work in this area includes [64] by Zhang *et al.*, which uncovered and categorized many novel client-side techniques by forcing the execution of phishing pages to trigger the cloaking behavior. Acharya *et al.* in [12] uncovered that phishing pages can successfully evade blocklists by knowing how to identify their crawlers, and Oest *et al.* in [44] demonstrated that cloaking from non-mobile based devices as a phishing page can ensure that attacker's page goes unmarked by the blocklists for more than 48 hours.

Kondracki *et al.* in [32] uncovered a massive blindspot of the phishing detection ecosystem that was Man-in-the-middle phishing kits. Kits that would transparently forward the victim's connection to the target page, mimicking brand logos on pages like Outlook without any configuration. Fortunately, the authors addressed the blind spot by demonstrating that these proxies remain fingerprintable using TLS fingerprinting. [62] proposed a similar attack, however, one that used JavaScript and NoVNC to trick the user into signing into their account through a VNC session in their browser.

With adversaries becoming creative with their evasions and obfuscation techniques, some novel defenses have also opted to think outside the box. Zhang *et al.* in [66] proposed a phishing defense solution that leverages the high likelihood of a phishing page cloaking away from a crawler to the defender's advantage. They demonstrated that a web browser configured to look like a crawler triggers a cloaking response from phishing pages, ensuring that victims never see the page while maintaining compatibility with all of Alexa's Top One Million websites. Meanwhile, using CAPTCHAs [61] utilized vision-based models to combat phishing pages.

To better understand why, other than cloaking, phishing pages may choose to fingerprint, Lin *et al.* in [36] showed that browser fingerprints could be successfully used to bypass multi-factor authentication, a system meant to be a last line of defense against stolen credentials, for 10 out of 16 websites that provide popular services.

Phishing kits: Much can be studied about the phishing ecosystem via phishing kits. Cova *et al.* in [20] uncovered that most "free" phishing kits contain a backdoor, effectively serving as a way to offload the deployment of a campaign to a 3rd party while siphoning off their stolen credentials.

Similar to our goals, PhishKitA [16] uses a dataset of phishing kits gathered through *KitPhisher* and a collection of features extracted from the HTML DOM to classify websites into their matching kit. They achieved an F1 score 0.91 when classifying 2,000 pages (1,141 benign and 859 phishing) from features extracted from their kit. However, their multi-class classifier for identifying the kit only achieved an F1 score of 0.39. Merlo *et al.* in [40] further expanded on our understanding of phishing kit lineages by looking at over 20,000 phishing kits and identifying, via token similarity, most of them as clones of one another or previously encountered kits. Prominent work in extending our understanding of phishing attacks includes Han *et al.* in [27], where they monitor the deployment of phishing kits by adversaries that compromise vulnerable web servers by hosting a well-sandboxed honeypot. They collected 643 phishing kits and established that kits take minutes to install and test and can remain undetected for weeks. Using these kits, they were also able to identify evasion techniques used by these kits, like path randomization per visit, which back then was enough to bypass Google SafeBrowsing.

In [45], Oest *et al.* manually analyzed phishing kits to establish the taxonomy for server-side cloaking, and in [14], Bijmans *et al.*, after collecting phishing kits by watching TLS transparency logs to identify Dutch brank phishing domains, manually created a fingerprint from static features to analyze their prevalence in the wild.

More recently, Lee *et al.* in [34] provides a server-side script (PHP) level analysis of phishing kits, finding that dynamically generated URLs are still standard in the ecosystem and observing seasonality in the kits they were able to obtain.
Extending the understanding of the phishing ecosystem:

Similar to our methods, Rola *et al.* in [53] deployed a modified Chromium browser to gather data and analyze phishing website browser APIs utilizing a pre-selected API list focused on first and third-party scripts for phishing pages. They find that the majority of the most visited phishing pages (identified via browser telemetry data) deploy fingerprinting scripts, sometimes varying from the ones of the original brand they portray. At the same time, they accessed this at a script level, reinforcing our finding that phishing pages vary vastly from their original page.

Oest *et al.* in [46] demonstrates the full lifecycle of a phishing campaign by employing the fact that phishing pages often copy assets from the target domain and refer the victim back to the original page afterward. By collaborating with a significant financial institution, they developed a framework for leveraging this data to track a phishing page from its deployment to blocklists flagging the page as phishing. [46] observed all techniques highlighted by prior work: cloaking, user-specific URL generation, man-in-the-middle proxies, and short-lived bursty attacks. Expanding our understanding of the victim experience on a phishing website, Subramani *et al.* in [59] developed a crawler.

7.2 Dynamic analysis of webpages

Our work shares a methodology for dynamic analysis enabled by web measurement frameworks like OpenWPM [22] and VisibleV8 [30]. Su *et al.* used VisibleV8 traces and taint analysis in [58] to discover emerging fingerprinting techniques. Sarker *et al.* used VisibleV8 to create an oracle for detecting obfuscation [54]. Such an oracle was made possible by the observation that VisibleV8 marks the execution of an API at a given source line. At the same time, obfuscation techniques ensure that the API is not textually available there. And Pantelaios *et al.* used a combination of VisibleV8 and force execution modifications to the Chromium engine to identify and defeat JavaScript evasion techniques while also leveraging API traces and clustering to identify previously unlabeled malicious extensions [49]. Iqbal *et al.* used OpenWPM to capture execution traces from tranco top 100K URLs of Tranco, training a classifier on a mixture of dynamic and static features extracted from the JavaScript's AST and execution traces, respectively, to achieve a 99.8% accuracy in identifying fingerprinting scripts online.

Our work differs from prior work in multiple ways. To date, we are the most successful in identifying phishing kits via the page's dynamic features; moreover, our differentiation is entirely automated, requiring no prior rule-based identification of kits. While we integrate many of the techniques annotated from prior work, we contribute an up-to-date understanding of their distribution in their ecosystem, and we do so at the cluster level, which we argue is more likely to control for multiple deployments of the same sophisticated kit.

8 Limitations and Future Work

Incorrect Ground truth mapping: While we manually validate our construction of kit families, we do not verify if the kit acquired is the kit deployed on the page. This remains an unexplored problem in the literature. Prior work has focused on studying the kits obtained from these sources, not necessarily validating that these kits were the ones deployed. **Unexplored Page states**: While we visit and loiter on phishing pages, we do not explore and interact with the pages. With recent work in LLM-powered crawlers, and ML-guided browser automation [59], we leave this to future work to use to collect a more complete list of browser APIs executed by a phishing page, at all stages of the phishing page. **Non-clustered pages**: For this paper, we ignored any page that did not execute at least eight distinct browser APIs in a first-party context. 376,762 pages did not execute any JavaScript at all, meaning that they are limited to server-side cloaking, and dynamic page generation on the server-side for obfuscating static features, while a possibility, we consider server-side techniques out of scope for our research purposes. Most of the chunk of pages were disqualified for not executing enough browser APIs or executing many of them in a first-party context. Including all party scripts would have muddied the similarities with shared usage of standard libraries, which has been documented in prior work [35], and allowing executions from redirected pages could introduce features into our clustering from the benign pages they are redirecting to. Phishing pages that employ server-side cloaking could redirect the victim to a benign page that loads scripts in a first-party context relative to itself; however, this causes pages to be grouped based on the redirection target. This can be an avenue for future work to explore, as including first-party scripts relative to the origin (current page loaded) would become more valuable if the crawler is automated to interact with the phishing page, to extract behaviors.

9 Conclusion

In this paper, we provided a workflow for researchers and analysts to automatically differentiate between a collection of phishing pages, based on a common underlying kit or shared techniques, if the behaviors are too generic. We show an accuracy of 97% against a dataset of pages and kits collected from the wild. With a curated mapping of techniques to browser APIs and 744,903 pages in which we identify *15,063* clusters, we explore what techniques are universal, widespread across kits, or kit-specific.

10 Ethical consideration

This research relied on publicly and commercially available URLs detected using proprietary methods to be phishing websites. No additional threat intelligence is attached to the URLs we are planning to release, and to ensure that any confidential information is not accidentally leaked through the URLs (like GET parameters that are indications of a test submission before the URL was submitted to the feeds), we blind the GET parameters of the URLs we visited upon release.

We did not collect or store any identifiable information about the individuals behind phishing kits or the pages, and we did not conduct any live testing of their networks, as we only visited at most twice upon ingestion.

11 Open Science policy

We support our dataset being used for any follow-up study of phishing ecosystem or reproducibility work. Our datasets are available for vetted researchers *upon request*. Releasing the phishing kits to be available to the public poses a security risk, as it would make these ready-to-deploy phishing pages available for anyone to modify and use. We also observed names, addresses, card numbers, and IP addresses in assets of phishing kits we have collected, which could effectively be used to identify prior victims of the pages.

References

[1] APWG | Phishing Activity Trends Reports. https://apwg.org/trendsreports/.

[2] cybercdh/kitphishr: A tool designed to hunt for phishing kit source code. `https://github.com/cybercdh/kitphishr`.

[3] Introducing quieter permission UI for notifications. `https://blog.chromium.org/2020/01/introducing-quieter-permission-ui-for.html`.

[4] Latest Alerts and Advisories | NJCCIC. `https://www.cyber.nj.gov/Home/Components/News/News/1424/214`.

[5] Recognize and Report Phishing | CISA.

[6] Sophisticated Spearphishing Campaign Targets Government Organizations, IGOs, and NGOs | CISA. `https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-148a`.

[7] Threat Actor Leverages Compromised Account of Former Employee to Access State Government Organization | CISA. `https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-046a`.

[8] Tycoon2FA New Evasion Technique for 2025. `https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/tycoon2fa-new-evasion-technique-for-2025/?hs_amp=true`.

[9] Eval() - JavaScript | MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval, January 2025.

[10] Unit 42. Threat Actor Groups Tracked by Palo Alto Networks Unit 42. `https://unit42.paloaltonetworks.com/threat-actor-groups-tracked-by-palo-alto-networks-uni` June 2024.

[11] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, pages 1681–1698, New York, NY, USA, November 2020. Association for Computing Machinery.

[12] Bhupendra Acharya and Phani Vadrevu. {PhishPrint}: Evading Phishing Detection Crawlers by Prior Profiling. pages 3775–3792.

[13] Anti-Phishing Working Group (APWG). eCrime Exchange (eCX). `https://apwg.org/`, 2025.

[14] Hugo Bijmans, Tim 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 (USENIX Security 21)*, pages 3757–3774, 2021.

[15] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[16] Felipe Castaño, Eduardo Fidalgo Fernañdez, Rocío Alaiz-Rodríguez, and Enrique Alegre. PhiKitA: Phishing Kit Attacks Dataset for Phishing Websites Identification. *IEEE Access*, 11:40779–40789, 2023.

[17] SANS Internet Storm Center. USPS Phishing Scam Targeting iOS Users. https://isc.sans.edu/diary/30078.

[18] Daiki Chiba, Hiroki Nakano, and Takashi Koide. DomainLynx: Leveraging Large Language Models for Enhanced Domain Squatting Detection. *ArXiv*, abs/2410.02095, 2024.

[19] Cisco Talos Intelligence Group (Talos). Phishtank. https://phishtank.org/, 2025.

[20] Marco Cova, Christopher Kruegel, and Giovanni Vigna. There is No Free Phish: An Analysis of "Free" and Live Phishing Kits.

[21] Dinil Mon Divakaran and Adam Oest. Phishing Detection Leveraging Machine Learning and Deep Learning: A Review. *IEEE Security & Privacy*, 20(5):86–95, September 2022.

[22] Steven Englehardt and Arvind Narayanan. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of ACM CCS 2016*, 2016.

[23] Edward B Fowlkes and Colin L Mallows. A Method for Comparing Two Hierarchical Clusterings. *Journal of the American statistical association*, 78(383):553–569, 1983.

[24] Yanick Fratantonio, Luca Invernizzi, Loua Farah, Kurt Thomas, Marina Zhang, Ange Albertini, Francois Galilee, Giancarlo Metitieri, Julien Cretin, Alexandre Petit-Bianco, David Tao, and Elie Bursztein. Magika: AI-Powered Content-Type Detection. In *Proceedings of the International Conference on Software Engineering (ICSE)*, April 2025.

[25] Google Inc. Catapult. https://chromium.googlesource.com/catapult/, 2025.

[26] Google Inc. Puppeteer. https://pptr.dev/, 2025.

[27] Xiao Han, Nizar Kheir, and Davide Balzarotti. PhishEye: Live Monitoring of Sandboxed Phishing Kits. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1402–1413, New York, NY, USA, October 2016. Association for Computing Machinery.

[28] Microsoft Threat Intelligence. New sophisticated email-based attack from NOBELIUM.

[29] Microsoft Threat Intelligence. Franken-phish: TodayZoo built from other phishing kits. https://www.microsoft.com/en-us/security/blog/2021/10/21/franken-phish-todayzoo-built-from-other-phishing-kits/, October 2021.

[30] Jordan Jueckstock and Alexandros Kapravelos. VisibleV8: In-browser Monitoring of JavaScript in the Wild. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2019.

[31] Koide, Takashi and Fukushi, Naoki and Nakano, Hiroki and Chiba, Daiki. Phishreplicant: A language model-based approach to detect generated squatting domain names. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, page 1–13, New York, NY, USA, 2023. Association for Computing Machinery.

[32] Brian Kondracki, Babak Amin Azad, Oleksii Starov, and Nick Nikiforakis. Catching Transparent Phish: Analyzing and Detecting MITM Phishing Toolkits. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 36–50, New York, NY, USA, November 2021. Association for Computing Machinery.

[33] Hung Le, Quang Pham, Doyen Sahoo, and Steven C. H. Hoi. URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection. *ArXiv*, abs/1802.03162, 2018.

[34] Woonghee Lee, Junbeom Hur, and Doowon Kim. Beneath the phishing scripts: A script-level analysis of phishing kits and their impact on real-world phishing websites. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 856–872. ACM.

[35] Kyungchan Lim, Jaehwan Park, and Doowon Kim. Phishing Vs. Legit: Comparative Analysis of Client-Side Resources of Phishing and Target Brand Websites. In *Proceedings of the ACM Web Conference 2024*, WWW '24, New York, NY, USA, 2024. Association for Computing Machinery.

[36] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. Phish in Sheep's Clothing: Exploring the Authentication Pitfalls of Browser Fingerprinting. pages 1651–1668.

[37] 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. pages 1633–1650.

[38] Ruofan Liu, Yun Lin, Yifan Zhang, Penn Han Lee, and Jin Song Dong. Knowledge expansion and counterfactual interaction for {Reference-Based} phishing detection. pages 4139–4156.

[39] Heather McCalley, Brad Wardman, and Gary Warner. Analysis of Back-Doored Phishing Kits. In Gilbert Peterson and Sujeet Shenoi, editors, *Advances in Digital Forensics VII*, pages 155–168, Berlin, Heidelberg, 2011. Springer.

[40] Ettore Merlo, Mathieu Margier, Guy-Vincent Jourdan, and Iosif-Viorel Onut. Phishing kits source code similarity distribution: A case study. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 983–994. ISSN: 1534-5351.

[41] Mitchell Krog and Nissar Chababy. PhishingDB. https://github.com/Phishing-Database/Phishing.Database, 2025.

[42] Mozilla. Restricting Notification Permission Prompts in Firefox. https://blog.mozilla.org/futurereleases/2019/11/04/restricting-notification-permission-prompts-in-firefox, November 2019.

[43] Aleksandr Nahapetyan, Sathvik Prasad, Kevin Childs, Adam Oest, Yeganeh Ladwig, Alexandros Kapravelos, and Bradley Reaves. On SMS Phishing Tactics and Infrastructure. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 169–169. IEEE Computer Society, 2024.

[44] Adam Oest, Yeganeh Safaei, Adam Doupe, 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)*, pages 1344–1361, San Francisco, CA, USA, May 2019. IEEE.

[45] Adam Oest, Yeganeh Safei, Adam Doupe, Gail-Joon Ahn, Brad Wardman, and Gary Warner. Inside a Phisher's Mind: Understanding the Anti-Phishing Ecosystem through Phishing Kit Analysis. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–12, San Diego, CA, May 2018. IEEE.

[46] 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. pages 361–377.

[47] Berstend on Github. Puppeteer stealth plugin. https://www.npmjs.com/package/puppeteer-extra-plugin-stealth, 2025.

[48] OpenPhish. OpenPhish. https://www.openphish.com/, 2025.

[49] Nikolaos Pantelaios and Alexandros Kapravelos. FV8: A Forced Execution JavaScript Engine for Detecting Evasive Techniques. In *Proceedings of the USENIX Security Symposium*, August 2024.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[51] Ch. Chakradhara Rao, A.V.T. Raghav Ramana, and B. Sowmya. Detection of phishing websites using hybrid model. 2018.

[52] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Conference on Empirical Methods in Natural Language Processing*, 2007.

[53] Iskander Sanchez-Rola, Leyla Bilge, Davide Balzarotti, Armin Buescher, and Petros Efstathopoulos. Rods with Laser Beams: Understanding Browser Fingerprinting on Phishing Pages. pages 4157–4173.

[54] Shaown Sarker, Jordan Jueckstock, and Alexandros Kapravelos. Hiding in Plain Site: Detecting JavaScript Obfuscation through Concealed Browser API Usage. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, page 648–661, October 2020.

[55] SecurityTrails. Urlscan. https://urlscan.io/, 2025.

[56] Hossein Shirazi, Bruhadeshwar Bezawada, and Indrakshi Ray. "kn0w thy doma1n name": Unbiased phishing detection using domain name based features. *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, 2018.

[57] Ben Stock, Benjamin Livshits, and Benjamin Zorn. Kizzle: A signature compiler for detecting exploit kits. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 455–466, 2016.

[58] Junhua Su and Alexandros Kapravelos. Automatic Discovery of Emerging Browser Fingerprinting Techniques. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 2178–2188, New York, NY, USA, 2023. Association for Computing Machinery.

[59] 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 '22, pages 589–604. Association for Computing Machinery.

[60] Bhaskar Tejaswi, Nayanamana Samarasinghe, Sajjad Pourali, Mohammad Mannan, and Amr Youssef. Leaky Kits: The Increased Risk of Data Exposure from Phishing Kits. In *2022 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–13, Boston, MA, USA, November 2022. IEEE.

[61] Xiwen Teoh, Yun Lin, Ruofan Liu, Zhiyong Huang, and Jin Song Dong. {PhishDecloaker}: Detecting {CAPTCHA-cloaked} phishing websites via hybrid vision-based interactive models. pages 505–522.

[62] Jonas Tzschoppe and Hans Löhr. Browser-in-the-middle - evaluation of a modern approach to phishing. In *Proceedings of the 16th European Workshop on System Security*, EUROSEC '23, pages 15–20. Association for Computing Machinery.

[63] Rakesh M. Verma and Avisha Das. What's in a url: Fast feature extraction and malicious url detection. *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*, 2017.

[64] 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. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1109–1124, San Francisco, CA, USA, May 2021. IEEE.

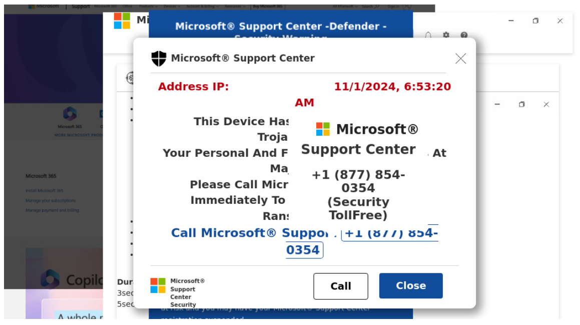
[65] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kapravelos, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. CrawlPhish: Large-scale analysis of client-side cloaking techniques in phishing. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1109–1124. ISSN: 2375-1207.

[66] 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 '22, pages 3165–3179, New York, NY, USA, November 2022. Association for Computing Machinery.

Appendix A

Table 5: Summary of all API calls that match to a category identified by Crawlphish

Category	Clusters	Pages
User-Interaction		
Pop-up (Total)	104	1,323
Accelerometer	7	63
Clipboard.readText	1	2
Geolocation.getCurrentPosition	55	479
Gyroscope	4	10
MediaDevices.getUserMedia	5	55
Notification.requestPermission	16	148
Window.alert	22	581
Mouse	35	62
DomEvents	10,402	362,700
Fingerprint		
Total	9,131	289,014
Navigator.userAgent	8,641	266,126
HTMLDocument.cookie	4,664	124,110
HTMLDocument.referrer	2,175	39,344
Bot Detection		
Timing	2,528	78,202


Microsoft® Support Center -Defender -

Microsoft® Support Center

Address IP: 11/1/2024, 6:53:20 AM

This Device Has Troja

Your Personal And F At Ma

Please Call Micr Immediately To Rans

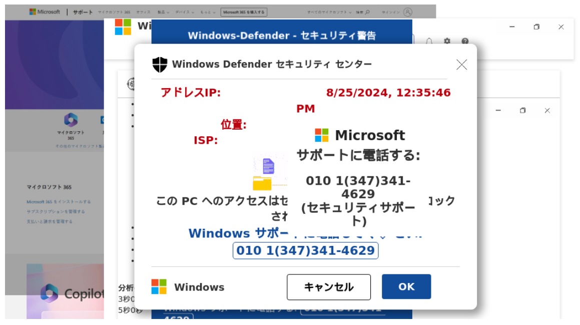
Microsoft® Support Center

+1 (877) 854-0354 (Security TollFree)

Call Microsoft® Suppo +1 (877) 854-0354

Microsoft® Support Center Security

(a) Page variant in English


Windows-Defender - セキュリティ警告

Windows Defender セキュリティ センター

アドレスIP: 8/25/2024, 12:35:46 PM

位置:

ISP:

Microsoft サポートに電話する:

010 1(347)341-4629 (セキュリティサポート)

この PC へのアクセスはセ ロック さ

Windows サポー

010 1(347)341-4629

Windows

(b) Page variant in Japanese

Figure 10: Cropped screenshots from Cluster-53d5c420, IP addresses and location redacted to ensure anonymity of the authors.

Table 6: Manual mapping of phishing techniques to browser APIs

Technique	Category	Identifying markers
Fingerprinting extraction	Credential Harvesting	10 Fingerprinting API calls and an exfiltration related API call from [58]
Client-side IP check	Evasion	`Window.fetch` `XMLHttpRequest.open`
Timing bot detection	Evasion	`Performance.now + Timeout`
Encryption	Obfuscation	`Subtlecrypto.decypt`
Encoding	Obfuscation	`TextDecoder.decode` `window.atob`
Dynamic script Evaluation	Obfuscation	`eval`

Basic fingerprinting	Evasion	`HTMLDocument.cookie`
		`HTMLDocument.referrer`
		`Navigator.userAgent`

Dynamic script creation	Evasion	`HTMLScriptElement.text` `HTMLScriptElement.innerHTML`
Cloudflare Turnstiles	Evasion	`Window.turnstile`

Pop-ups	Evasion	`Navigator.requestMIDIAccess`
		`Clipboard.readText`
		`Geolocation.getCurrentPosition`
		`MediaDevices.getDisplayMedia`
		`HID.requestDevice`
		`Window.{confirm|alert|prompt}`
		`Accelerometer|Gyroscope`
		`Window.showModalDialog`
		`MediaDevices.getUserMedia`
		`SyncManager.register`
		`Clipboard.read`
		`Serial.requestPort`
		`USB.requestDevice`
		`Window.queryLocalFonts`
		`Notification.requestPermission`