Inside the Phishing Reel: Leveraging Browser Instrumentation to Analyse Evasive Phishing

João Pedro Favoretti[1],**Fernando Dantas**[2],**Lourenço Alves Pereira Jr**[1]

[1] Instituto Tecnológico de Aeronáutica – ITA

[2]Microsoft Security Research

{favoretti,ljr}@ita.br, fernando.dantas@microsoft.com

Abstract. *Phishing websites remain a persistent threat, compromising millions of user credentials each year. While advanced classifiers have improved detection accuracy, they rely on crawlers that are easily redirected by cloaking techniques, limiting their effectiveness in real-world deployments. In this work, we propose a novel data collection approach using a modified Chromium browser capable of collecting information from phishing samples despite client-server cloaking tactics. Using this approach, we collected 432,237 unique phishing samples over eight months and introduced a clustering algorithm that groups samples based on shared deployment characteristics, achieving a Normalized Mutual Information (NMI) score of 0.846.*

1. Introduction

In the fourth quarter of 2024, on average more than 300.000 unique monthly phishing attacks were reported by [APWG 2024]. In addition to hurting the image of well-established brands and leaking private information from users (i.e. identity information and credentials), attackers annually cause indirect financial damage to the security sector of companies due to the need for defensive infrastructures that are used to protect themselves and clients, as reported in [Bitaab et al. 2020]. Even worse, a few studies done by [Oest et al. 2018], and [Oest et al. 2020b] demonstrated that phishing kits, which are sold as a ready-to-use malicious template, continue to evolve and are being deployed due to their automatic deployment mechanisms, as shown by [Oest et al. 2019], resulting in a constant cat-and-mouse game between phishing attacks and defense mechanisms.

As pointed out by [MSDR 2024], [Oest et al. 2020b] and [Thomas et al. 2017] observations show that phishing still remains a popular social engineering attack among cybercriminals due to its low entry level prices even for complex and highly evasive systems, supported by illegal underground services that were exposed by [Hao et al. 2013]. Today, phishing blacklisting tools such as Google Safe Browsing and Microsoft Defender SmartScreen are being used to prevent users from accessing phishing websites by using a huge list of blocked URLs. However, as stated by [Oest et al. 2020b], those tools rely on a backend service to verify if a reported URL is indeed malicious. Therefore, automatic detection tools are required to classify phishing websites faster, reducing the Golden Hour that attackers have to exploit users, as stated by [Oest et al. 2020b].

Fortunately, prior studies such as [Tian et al. 2018], [Lee et al. 2020], and [Liu et al. 2024] have introduced supervised classification tools that leverage website appearance, URL structures, and content-based features to detect phishing attempts, achieving promising levels of accuracy. However, their effectiveness is significantly challenged

by more evasive phishing strategies. [Oest et al. 2018] and [Oest et al. 2020a] have shown that phishing actors increasingly deploy cloaking mechanisms to hide malicious content from security analysts and automated crawlers, primarily through fingerprinting techniques that detect the environment in which the website is being accessed. These mechanisms allow phishing pages to selectively serve benign content when accessed by known security infrastructures, thus evading detection. [Zhang et al. 2021] further demonstrated the use of sophisticated client-side fingerprinting to analyze browser characteristics before executing malicious redirections, providing a detailed enumeration of techniques observed in the wild. As a result, phishing samples employing client-server evasion are often excluded from classification datasets due to their inability to crawl and extract features. This exclusion not only undermines the robustness of detection models but also creates a disconnect between experimental research and the complex realities of phishing in the wild.

To address the challenge posed by evasive phishing pages that employ JavaScript-based fingerprinting, followed by client- or server-side redirection to conceal their malicious content, we leverage the insight that such advanced samples still execute code, even when they display no visible signs of phishing content to automated crawlers. This enables the extraction of valuable behavioral information from these pages. Several prior efforts have explored how websites interact with JavaScript APIs to perform fingerprinting, such as [Jueckstock and Kapravelos 2019] and [Sanchez-Rola et al. 2023], while lighter-weight systems like [Roesner et al. 2012] have attempted to inject in-band hooks into the target application's namespace. However, as noted by [Jueckstock and Kapravelos 2019], these in-band approaches suffer from limited coverage and are vulnerable to detection. Alternative strategies using the Chrome Debugging Protocol have also been investigated by [Sanchez-Rola et al. 2023] and [Dambra et al. 2022] to identify fingerprinting behaviors during page execution. Yet, such methods lack support for analyzing post-interaction redirections—an essential element in many phishing infrastructures, as highlighted by [Oest et al. 2018]. In contrast, out-of-band browser instrumentation systems, like those proposed by [Jueckstock and Kapravelos 2019] and [Li et al. 2018], allow for tracing JavaScript object usage across the full execution of a webpage. Building on this, our work employs an out-of-band system that traces V8 bytecode execution in a custom browser, enabling both the collection of detailed execution traces and the ability to follow subsequent page redirections, thereby capturing the behavior of phishing samples despite containing evasion code.

Building on prior work, we leverage execution trace features extracted from JavaScript resources to cluster phishing samples and identify groups that originate from the same phishing kit. This approach draws inspiration from [Bijmans et al. 2021], who used source code similarity to fingerprint phishing kits and track their deployment via CertStream, and from [Sanchez-Rola et al. 2023], who explored clustering based on execution traces related only to fingerprinting behavior. Additionally, [Kondracki et al. 2021] analyzed Man-in-the-Middle (MiTM) phishing toolkits that function as reverse proxies, highlighting the diversity of phishing infrastructure. In contrast to these works, which either rely on source code similarity or focus on specific types of phishing kits, our method generalizes the clustering process by using behavioral features derived from JavaScript execution. We then apply this clustering to real-world phishing samples collected over a defined time period, allowing us to analyze the deployment and activity curves of distinct

phishing kits as they appear in the wild.

In this paper, we propose a new approach to collect features from phishing samples that are robust to client-side evasion techniques in a way that allows us to use more samples than state-of-the-art methods. To do so, we leverage browser instrumentation techniques to access JavaScript bytecode execution traces from the Chromium browser. Then, we use this custom browser to deploy a crawler to collect information for almost eight months, with more than 400.000 unique phishing samples. Then, we develop a clustering algorithm that leverages those collected features in a way that is more generic than previous approaches, obtaining 0,846 NMI score in a controlled and manually inspected dataset of phishing kit deployments. Finally, we use the clustering algorithm to develop a day-by-day analysis on our samples to discover what is the number of phishing samples deployed in the wild that were already reported before. To enhance the combat of phishing both in academic and industry research, we made the toolkit developed to process this work completely available at Github[1]. Our contributions are, thus, the following:

- We propose a more embracing method to collect data from phishing samples that is not prone to evasion.
- We deploy a scalable pipeline to collect phishing samples using a specialized browser.
- We create a clustering algorithm that uses the collected data to group phishing samples into groups that represent redeployments of the same phishing kit.
- We leverage the clustering algorithm to observe how phishing kits are reused daily.

The remainder of this paper is organized as follows. Section 2 provides an overview of the state of the art in the domain of phishing, browser tracing, and phishing clustering. Then, Section 3 introduces the new framework created to obtain phishing data, cluster and analyse phishing kit usage through time. Section 4 reports the preliminary results we had working with this data and how we can use this even further. Finally, we conclude all the experiments and present further perspectives in Section 5.

2. Related Works

Phishing. Several works have addressed critical aspects of phishing detection and analysis. For instance, [Zhang et al. 2021] developed a comprehensive framework for the automated crawling of phishing websites, focusing on client-side cloaking techniques and proposing a classifier based on simple fingerprinting features. Similarly, [Acharya and Vadrevu 2021] introduced an innovative method to analyze and compare crawler behaviors used by blacklisting tools, demonstrating how fingerprinting can help phishing sites evade detection. In another approach, [Tian et al. 2018] combined content- and domain-based features in a classifier to detect phishing pages, while explicitly accounting for common evasion techniques. Moreover, [Liu et al. 2022] modeled webpages using visual diagrams of structural elements to distinguish malicious from benign sites. Building upon this work, [Liu et al. 2024] proposed PhishLLM, a novel LLM-based system that detects phishing by inferring brand-domain relationships and evaluating credential-stealing intent. Additionally, [Kondracki et al. 2021] explored phishing toolkits that act as reverse proxies in man-in-the-middle attacks, capturing sensitive

[1]`https://github.com/joaofavoretti/Phishing-Kit-Analysis-Toolkit/`

data while mimicking legitimate sites. Despite these significant advances in phishing detection, many of the aforementioned studies disregard samples that exhibit client-server cloaking or resist automated crawling, often treating them as noise or outliers. In contrast, our work proposes a method to leverage precisely those hard-to-analyze samples, aiming to shed light on evasive phishing strategies that have been largely overlooked in prior research.

Browser Tracing. Prior work on browser tracing and instrumentation has laid important groundwork for understanding web-based threats. For example, [Jueckstock and Kapravelos 2019] introduced a browser instrumentation technique capable of logging JavaScript object accesses and function calls, demonstrating its utility in detecting bot-detection mechanisms through the analysis of property usage in benign websites. Similarly, [Xu et al. 2013] applied static analysis to identify JavaScript functions frequently employed by malware kits, thereby highlighting common malicious patterns. In a complementary direction, [Li et al. 2018] extended the Chromium browser with an out-of-band system that inspects communication between websites and servers, offering insights into hidden exchanges. Building on these advancements, our work leverages browser instrumentation to extract runtime information directly from the JavaScript execution pipeline. This approach enables us to gather critical evidence from phishing websites—even those that perform client-side cloaking or display no visible content to the user—thus addressing limitations in prior approaches that rely heavily on visual or structural cues.

Clustering. Several studies have explored clustering techniques to better understand and categorize phishing activities. In this sense, [Bijmans et al. 2021] introduced a graph-based community detection approach to group phishing kits based on source code similarity and used CertStream data to track their deployment in the wild. However, their method is limited by its reliance on unobfuscated source code. In a related effort, [Sanchez-Rola et al. 2023] conducted one of the most comprehensive analyses of phishing page fingerprints, uncovering geographical and behavioral patterns; they also performed limited clustering on a small subset of their data to illustrate specific case studies. Similarly, [Sakurai et al. 2020] focused on HTTPS-based phishing, proposing a clustering and classification method using TLS-related features—particularly domain-based regex patterns—to identify trends in phishing domains. Building on these works, our research presents a clustering approach grounded in a novel feature extraction technique. Moreover, it extends prior efforts by demonstrating how clustering results can directly support phishing kit tracking. We validate our method through a detailed 10-day analysis, thereby showcasing its practical utility in uncovering evolving phishing infrastructure.

In summary, our work diverges from prior efforts by focusing on phishing samples that are typically dismissed as noise—those that employ client-server cloaking or evade automated crawling. Unlike existing detection approaches that rely heavily on visual, structural, or static features, we extract runtime data through browser instrumentation to capture dynamic behaviors that static analysis often misses. Additionally, we propose a novel clustering method based on this dynamic data, enabling the tracking of phishing kits even in the presence of obfuscation or evasive techniques. Together, these contributions address critical blind spots in previous research and offer a more comprehensive understanding of modern phishing infrastructure.

3. Method

In this section, we detail the methodologies employed in constructing each component of our experimental framework. We begin by describing the modifications made to the Chromium browser to enable instrumentation and explain the types of execution data this customized browser can extract from phishing samples. Next, we present the manually curated phishing dataset used to establish a reliable ground truth and enable a direct comparison with baseline approaches. We then outline the design of our crawler, which leverages the instrumented browser to continuously collect execution traces from phishing URLs. Following this, we describe the clustering algorithm developed to showcase the utility of the extracted features. Finally, we introduce the Projector pipeline, a system built upon the clustering algorithm to continuously analyze incoming phishing samples and detect potential redeployments of previously observed phishing kits.

3.1. Instrumented Browser

To capture detailed information about JavaScript execution in phishing websites, we adopt an approach inspired by [Jueckstock and Kapravelos 2019], modifying the Chromium browser's bytecode generation pipeline and hooking into its runtime function call mechanism. This instrumentation allows us to log both JavaScript property accesses and function invocations. Unlike prior work such as [Sanchez-Rola et al. 2023], which relies on the Chrome DevTools Protocol (CDP) and is limited in handling multi-stage redirection chains, our method is resilient to multiple URL redirections. Furthermore, while [Jueckstock and Kapravelos 2019] implementation was based on Chromium version 115.0.5790.170, our system uses a more recent version, Chromium 124.0.6367.78, providing broader compatibility and access to newer browser features.

At a technical level, to log function calls, we hook into Chromium's runtime call handler, ensuring that a log entry is generated every time a JavaScript function is invoked. To maintain full visibility, we disable TurboFan's function call optimization, which would otherwise bypass the runtime handler after initial execution. For property access logging, we instrument the bytecode generation process by injecting a custom bytecode instruction that triggers a logging function whenever a property is loaded or stored. We also monitor the browser's origin property from the global object, logging any changes internally to help identify redirections between different origins. Additionally, our instrumentation enables us to associate each executed instruction with its originating script file, making it possible to reconstruct the sequence of operations for specific JavaScript sources. In Listing 1, we exhibit the traces executed by a phishing sample that uses fingerprinting-related instructions, then redirects the user to the Facebook page. Even though our crawler got redirected, we were still able to capture information from the anti-bot script executed by the phishing instance, which allows us later to group samples that use the same anti-bot execution sequence. That result might indicate that the group of samples was deployed using the same phishing kit.

To support large-scale data collection, we containerize our custom instrumented version of Chromium using Docker, ensuring scalability and reproducibility. The resulting log format is structured such that for every process spawned by the browser when visiting a website, a separate log file is generated. These logs contain both the sequence of executed bytecode instructions and metadata about the corresponding source scripts

loaded during execution. Each instruction is explicitly linked to a script and its origin, enabling fine-grained behavioral analysis.

A key advantage of our approach is its robustness against code obfuscation, which is an increasingly common tactic in phishing kits. While obfuscation alters the appearance of source code, the actual sequence of JavaScript instructions executed by the browser remains consistent. By tracing execution at the bytecode level, our system bypasses obfuscation entirely, allowing us to capture the underlying behavior of phishing pages regardless of their surface-level code transformations. However, this approach might fall behind when dealing with completely static phishing pages that do not execute any JavaScript code execution, even though they might be considered simpler and easier to analyze by the current state-of-the-art classification tools.

Listing 1. JavaScript traces collection of evasive sample

```
1  REDIRECT https://compte-online.duckdns.org/
2  LOAD https://compte-online.duckdns.org/
3  LOAD https://compte-online.duckdns.org/antibot.js
4  EXECUTE https://compte-online.duckdns.org/antibot.js
5  GET Navigator.permissions
6  GET Permissions.query
7  CALL Permissions.query
8  GET Window.navigator
9  GET Navigator.userAgent
10 GET Window.chrome
11 GET Navigator.plugins
12 GET PluginArray.length
13 GET Navigator.appVersion
14 GET Navigator.connection
15 GET NetworkInformation.rtt
16 GET Window.location
17 SET Location.href
18 GET PermissionStatus.state
19 REDIRECT https://www.facebook.com/
20 LOAD https://www.facebook.com/
21 EXECUTE https://www.facebook.com/
22 GET Window.requireLazy
23 GET Window.Env
```

3.2. Manually Verified Dataset

To evaluate the features and instruction sequences obtained through our instrumented browser, we constructed a manually verified dataset of phishing pages that can be locally deployed from phishing kits obtained via Phishunt[2]. This dataset serves as a benchmark for assessing the effectiveness of our clustering algorithm. To facilitate deployment, we developed a script that launches each phishing kit within an isolated Docker container, using an Apache PHP web server to locally host each phishing website.

Our data collection process began with the retrieval of 837 phishing kits from the Phishunt open platform, spanning the period from May 2020 to October 2024. Each kit was manually deployed and evaluated to determine its "deployability" and "validity." To streamline this process, we created a deployment and classification tool, which not only simplifies manual analysis but also supports future improvements in phishing kit processing workflows. Alongside verifying deployability, we examined each kit for the presence of server-side and client-side redirection mechanisms—information that is publicly available in our GitHub project repository.

Following this validation process, 409 phishing kits were identified as deployable and valid. These kits were then manually clustered based on source code similarity and

[2]`https://phishunt.io/`

landing page characteristics, providing a ground truth for assessing both the crawler's behavior and the performance of our clustering algorithm. This curated dataset enables rigorous evaluation while reflecting real-world phishing kit diversity.

3.3. Continuous Crawling

The continuous crawling system operates through a modular architecture initiated by a Trigger module, which executes daily, twice at uniformly random times to reduce temporal bias in sample collection. Upon execution, the Trigger sequentially activates three main components. First, the Downloader module accesses publicly available feeds of validated phishing URLs, including PhishTank, OpenPhish, and PhishStats. Newly retrieved URLs are deduplicated against a local database to avoid redundant crawling. Any previously unseen URLs are then passed to the Analyzer module, which performs dynamic analysis using our instrumented Chromium-based browser to collect and save trace information from each sample.

For each URL, a dedicated Docker container is launched to isolate the environment and prevent resource caching between samples. The instrumented browser executes for up to 60 seconds, capturing JavaScript bytecode traces and runtime behavior, while it remains open. In parallel, we run [Kitphishr], a tool designed to detect phishing kits hosted on the server. Although KitPhishR results are not used as features for clustering, they contribute to expanding the dataset of phishing kits curated from Phishunt.

Post-analysis, all artifacts related to a given URL are stored in a uniquely named directory based on the hash of its domain, ensuring consistent deduplication across crawling sessions. Due to storage constraints on the local server, the collected data is regularly uploaded to a structured Google Drive repository, where samples are organized by source, date, and time of collection. This dataset is made available to the research community to support further studies on phishing detection and analysis.

3.4. Clustering Algorithm

To evaluate the proposed feature representation, we designed a multi-step clustering algorithm. The first step ① involves extracting Instruction Blocks from the JavaScript execution traces recorded by the browser during URL visits. Each Instruction Block represents a contiguous sequence of executed instructions associated with a specific resource (e.g., a JavaScript file). This abstraction accounts for variability in the loading order of resources and enables the identification of structural similarity between phishing samples that share components but differ in overall layout or control flow. This is particularly useful when analyzing different versions of the same phishing kit, which may reuse backend logic (e.g., command-and-control communication) while modifying the frontend interface.

Each Instruction Block is then ② embedded into a fixed-length vector using the SBERT (Sentence-BERT) embedding model, producing a 1024-dimensional representation. We selected SBERT over alternatives such as DOC2VEC after empirical comparison, as it demonstrated superior grouping of similar samples and benefits from Attention mechanisms that capture broader contextual dependencies. Additionally, SBERT's pretrained architecture ensures both computational efficiency and robust feature encoding. As a result, each phishing sample is represented by a variable-sized matrix of shape $n \times F$, where n is the number of Instruction Blocks (i.e., resources) and $F = 1024$.

Since clustering requires a fixed pairwise distance metric, and the samples produce matrices of differing sizes, we developed a custom distance computation strategy. First, we flatten the n × F matrix of each sample into n individual F-dimensional embeddings. All such embeddings across samples are pooled ③ and clustered using DBSCAN ④, assigning each embedding an identifier that groups similar instruction blocks. We then convert ⑤ each sample's set of embeddings into a sequence of identifiers produced by the clustering result. To measure the distance between two samples, we compute a modified Levenshtein distance ⑥ between their identifier sequences, incorporating an inverse exponential cost function that places greater emphasis on early instruction blocks. The decay function is parameterized by an alpha value controlling the weight drop-off over the sequence length.

Finally, this distance metric is used to compute an $O(n^2)$ pairwise distance matrix between samples ⑦, which is then clustered using DBSCAN to yield the final groupings of phishing samples based on behavioral similarity. The complete clustering algorithm is demonstrated in Figure 1 for a better visualization of the explanation provided.

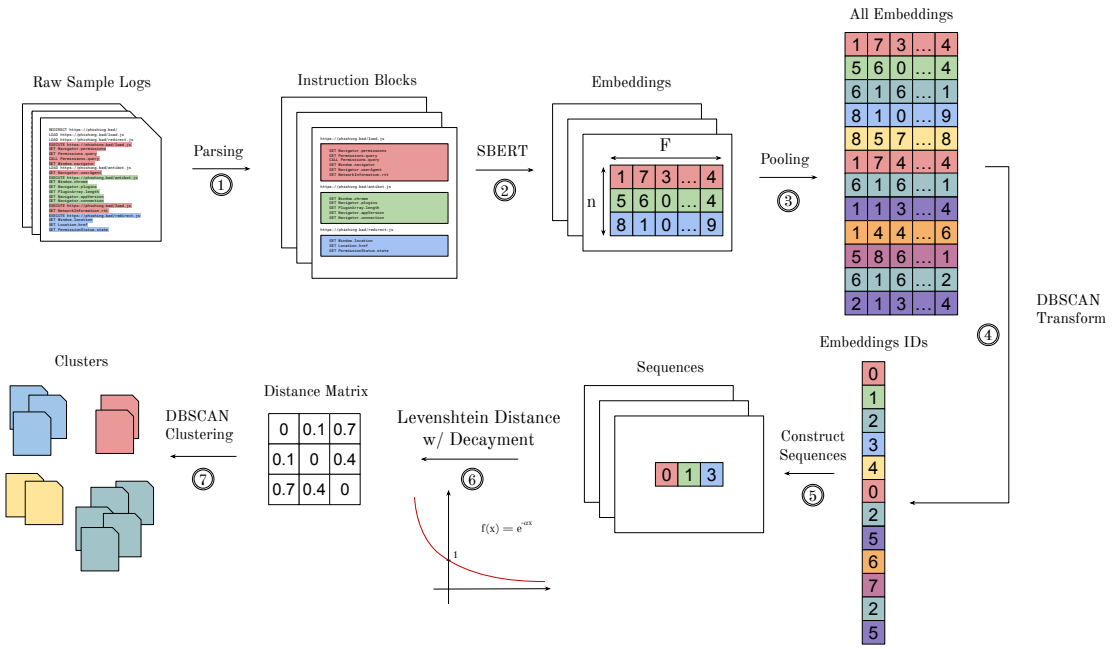

Figure 1. Clustering solution diagram.

3.5. Projector

Leveraging the continuous crawling infrastructure and the clustering algorithm described earlier, we introduce the Projector pipeline, which is a system designed to track the reuse and evolution of phishing kits over time. The central idea is to analyze samples collected each day in isolation and compare them against those collected in previous days. By projecting daily samples into a common geometric feature space defined by the embeddings of earlier samples, the pipeline enables direct comparison between new and historical phishing behaviors. As a result, the system can distinguish between novel phishing kits and redeployments of previously observed ones.

For each day, the Projector pipeline retrieves and processes the samples collected during prior periods, clusters both historical and new samples together, and then labels the current-day samples accordingly. A sample is considered new if it forms its own distinct cluster, while it is treated as a redeployment if it is assigned to an existing cluster containing previously seen samples.

To improve efficiency, we incorporated several optimizations into the Projector pipeline. One key optimization involves a sample uniqueness score used to reduce redundant processing. For each cluster, the algorithm computes the pairwise distances between samples and flags those that are near-identical to others in the same cluster. The uniqueness score is set to 1.0 for a representative sample, while duplicates are assigned a lower score—typically 0.0—allowing them to be ignored in downstream analyses. The logic is illustrated by the pseudocode in Listing 2, which iterates through each pair of samples in a cluster, comparing distances and retaining only one from a duplicated pair.

Listing 2. Uniqueness calculation for each cluster sample

```
uniqueness = [1] * len(cluster)
for i in range(0, len(cluster)):
    for j in range(i, len(cluster)):
        if uniqueness[j] > distances[i, j]:
            uniqueness[j] = distances[i, j]
            uniqueness[i] = 1.0
```

This filtering strategy reduces the number of redundant samples used in subsequent steps, resulting in an average reduction of 16.6% in processing load. Additionally, the pipeline caches clustering results from previous days, enabling incremental updates. This means only new samples, on average 1,232 per day, must be analyzed and clustered, significantly improving scalability and runtime efficiency.

4. Results

In this section, we present the results of our experiments, conducted using the methodologies described earlier. We begin by evaluating the system's ability to detect evasive phishing samples—specifically, those employing client-side fingerprinting followed by redirection—which are typically missed by prior state-of-the-art approaches. Next, we assess the consistency of our instrumented crawler through a three-part experiment, examining its behavior when repeatedly analyzing the same or similar phishing samples. We then report quantitative results obtained from a long-term crawling campaign, followed by an evaluation of the clustering algorithm's performance. As an extension, we demonstrate how the clustering method integrates with the Projector pipeline to detect repeated deployments of previously observed phishing kits. Finally, we present two baseline comparisons against recent work to highlight improvements in sample coverage and clustering effectiveness.

4.1. Cloaked Samples

In a preliminary experiment, to verify how many samples do express JavaScript traces while still redirecting the user to a benign page, we manually identified 32 phishing samples exhibiting fingerprinting behavior. These samples attempt to profile the user accessing the malicious page and subsequently redirect them to a benign domain. We sourced

these preliminary samples from crawled data collected in January 2025, after pre-filtering those that loaded resources from known benign domains. The list of benign domains was based on the Tranco Top 1M[3] list, updated in March 2025.

To automate the detection of such cloaking behavior and eliminate the need for manual analysis, we developed a natural language query[4] for the OpenAI o3-mini large language model (LLM). We tested this approach on a curated dataset of 48 samples, 32 of which performed fingerprinting prior to redirection, and 16 that redirected without fingerprinting.

Due to the token limitations of the LLM, we restricted the analysis to the first 3,000 instruction lines from each sample's execution log. This threshold is well below the model's maximum token capacity, ensuring sufficient space for the model's reasoning process. Using this setup, the model correctly classified 46 out of the 48 samples, achieving an accuracy of 95.8%. The two misclassified cases involved redirection events occurring beyond the 3,000-instruction limit, resulting in false negatives.

Applying this LLM-based analysis to a broader dataset of 1,377 unique samples from 2025 that loaded resources from benign domains, 701 samples were identified as using client-side fingerprinting followed by redirection to benign pages. This behavior suggests an intentional attempt to evade analysis tools or scrutiny by unwanted visitors, such as security analysts. Traditional phishing detection methods, which rely on access to the phishing landing page, would likely fail to detect these cloaked samples. In contrast, our instruction-level analysis enables the identification of such advanced evasion techniques employed by sophisticated phishing kits.

4.2. Crawler Behavior

To evaluate the consistency and behavior of our instrumented browser, we designed three distinct experiments. These experiments aim to assess the crawler's behavior when accessing the same page multiple times, and when interacting with multiple deployments of the same phishing kit.

In the first experiment, we examined the stability of crawled results by repeatedly visiting the same website. Specifically, we selected the top 3,000 domains from the Tranco 1M list and attempted to access their index pages to confirm their availability. After filtering out inaccessible or invalid pages, we obtained 1,657 accessible domains and selected the first 1,000 for the experiment. Each of these 1,000 pages was crawled three times, and we computed the Longest Common Subsequence (LCS) distance between the JavaScript instruction traces generated during each crawl. Excluding pages with no dynamic behavior (i.e., no JavaScript logs), the average pairwise LCS distance was 0.22. A manual inspection of 100 randomly selected samples confirmed that all differences were attributable to variations in the order of resource loading, which caused the associated JavaScript execution to be deferred or reordered—rather than indicating meaningful behavioral divergence.

The second experiment focused on live phishing pages. We manually selected six phishing URLs from the Phishtank database and re-crawled them after delays of 1

[3] `https://tranco-list.eu/`

[4] `https://github.com/joaofavoretti/Phishing-Kit-Analysis-Toolkit/blob/main/llm_parser/llm_parser.py`

and 5 minutes to assess potential temporal variation. All samples produced JavaScript instruction traces. In five of the six cases, the instruction logs were identical across time intervals. The single differing case again involved minor variation due to execution order changes stemming from browser resource-fetching behavior.

The third experiment addressed behavioral consistency across multiple deployments of the same phishing kit. We reused a manually labeled dataset containing clusters of visually and structurally similar phishing pages. Among 61 clusters analyzed, eight showed no differences in execution traces. For the remaining clusters, manual inspection revealed that observed differences were again due primarily to differences in script or resource loading order. However, we also observed minor naming differences between deployments. For instance, some pages accessed dynamic variables such as *window.jQuery224074545809895592341*, while others referenced *jQuery224034948906833229376l*, likely due to differing jQuery versions, without changes in phishing behavior.

One noteworthy case involved a phishing kit impersonating the Wells Fargo login page, originally observed in 2020. Among seven redeployments of this kit, six executed only a single JavaScript instruction after loading the login.php file. However, one anomalous sample executed a full Instruction Block, as shown in Listing 3, which includes a setTimeout call, DOM traversal, and form input modification. This difference illustrates how minor redeployments can occasionally introduce subtle execution path variations.

Listing 3. Execution sequence in anomalous redeployment sample

```
1 CALL Window.setTimeout
2 GET HTMLDocument.getElementsByTagName
3 SET HTMLInputElement.type
```

4.3. Crawling Results

Between September 9th and April 28th, we executed our crawler pipeline on phishing URLs sourced from three prominent platforms: PhishTank, PhishStats, and OpenPhish. Over this period, the crawler processed a total of 911,998 samples, corresponding to 432,237 unique URLs. The presence of duplicate samples arises from the same URL being reported multiple times within or across platforms.

In order to support further research in phishing detection and analysis, we have made both the crawled dataset and our collection infrastructure publicly available. In addition to the crawled samples, we extracted and open-sourced 848 phishing kits recovered during the crawling process. These kits can serve as a valuable resource for future studies involving local phishing kit deployment and behavioral analysis.

To contextualize the dataset, we include a time series illustrating the daily volume of phishing pages successfully crawled. Although our objective was to maintain continuous, daily operation of the pipeline, the crawler was hosted on a private server, which was occasionally subject to outages. On such days, URLs were retained and processed on the next available operational day. While this may introduce slight discrepancies in the crawl timestamps, such cases are isolated and can be treated separately in downstream analyses.

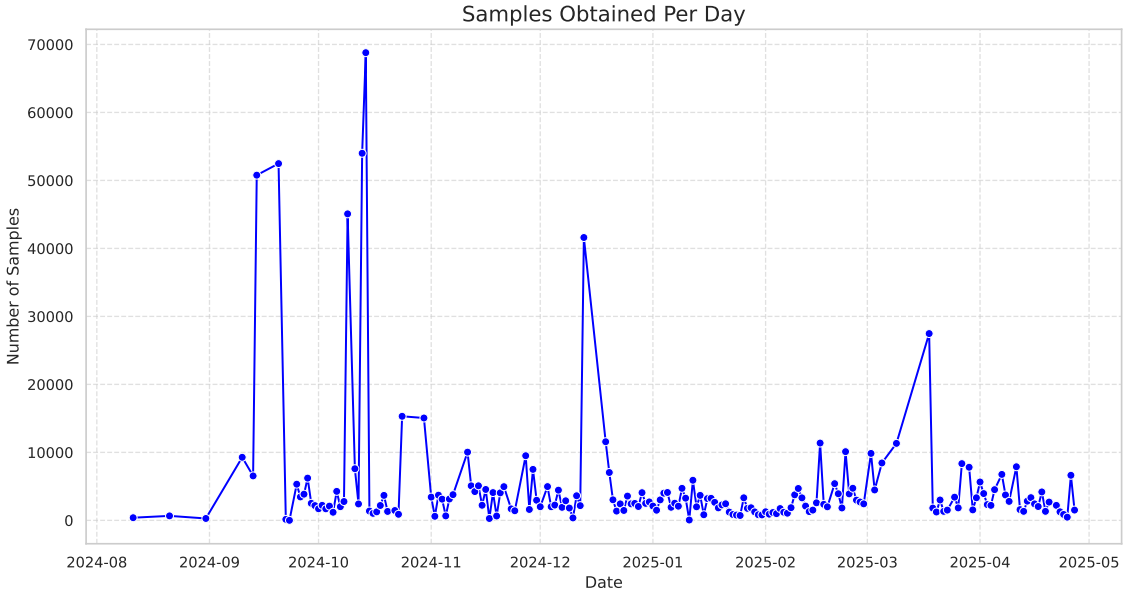

Figure 2. Amount of samples obtained per day.

4.4. Clustering Performance

To assess the effectiveness of the collected features, we developed a clustering algorithm aimed at grouping crawling results that likely represent different deployments of the same phishing kit. Our approach was evaluated using a manually labeled ground truth dataset. We performed a grid search over the algorithm's hyperparameters to identify the configuration that yielded the best clustering performance. For both DBSCAN algorithm usages, we tested six uniformly spaced values for the epsilon parameter within the range [0.1, 2.0], while we set the minimum number of samples per cluster to one. Additionally, for the matrix summarization step, we explored values for the *alpha* parameter in the range [0.1, 1.5]. The best configuration achieved a maximum Normalized Mutual Information (NMI) score of 0.846. We selected NMI as our evaluation metric due to its robustness in handling imbalanced cluster sizes, which is particularly important given the highly skewed nature of phishing kit deployments.

4.5. Generalization to Phishing Kits

Building on the strong clustering performance achieved by our unsupervised learning approach, we extended the analysis to phishing samples collected between September 26th and October 9th, 2024. The goal was to identify the largest clusters during this period and estimate the redeployment rate of phishing kits. In total, we analyzed 18,846 valid phishing samples, which were grouped into 1,433 clusters based on the feature distribution and clustering algorithm. Among these, 811 clusters were singletons—containing only one sample, indicating unique phishing pages. Conversely, three clusters contained over 1,000 samples each, and 19 clusters included more than 100 samples, suggesting widespread redeployments of certain kits, which is displayed in Figure 3. Monitoring the temporal evolution of these highly populated clusters over a longer period remains an important direction for future work.

We also investigated the daily redeployment rate—that is, the proportion of phishing samples that reappear across consecutive days as instances of previously observed

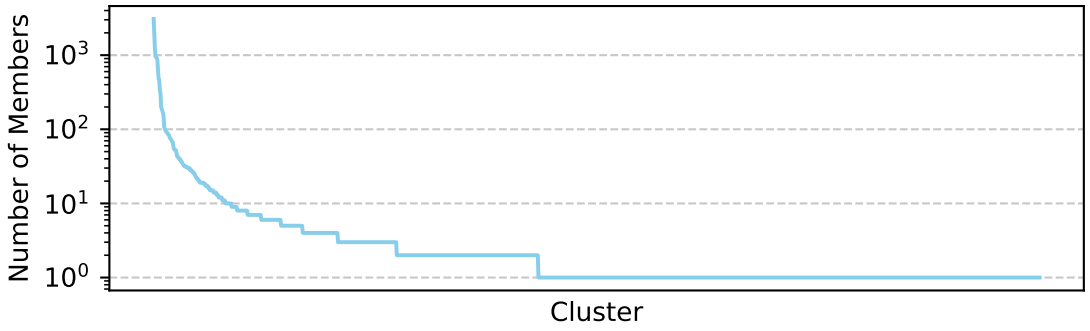

Figure 3. Cluster size distribution. Each point on the graph represents the size of an individual cluster, arranged in descending order.

kits. This information is particularly valuable for security analysts tasked with manually reviewing reported phishing pages. Our analysis revealed a consistent pattern, with an average redeployment rate of 90.29%, showed in Figure 4. This implies that 9 out of 10 phishing samples reported on a given day had already been observed in earlier days, and thus could potentially be preemptively blocked using execution trace-based signatures extracted by this work.

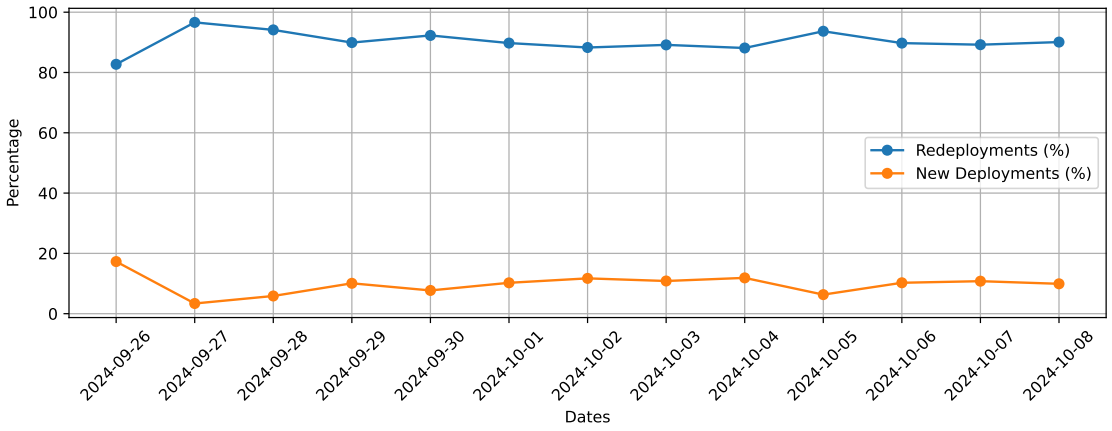

Figure 4. Redeployments and New Deployment rates obtained in the analyzed period of time.

4.6. Baseline: Sample Discard Rate

As a baseline evaluation, we compare our approach to existing state-of-the-art techniques in terms of sample coverage—that is, the proportion of phishing samples that can be successfully analyzed. Specifically, we reproduced the crawler from [Liu et al. 2024] and tested it against our controlled, self-hosted phishing dataset. The reproduced crawler excluded 178 out of 409 samples, resulting in a discard rate of 43.5%. These exclusions stemmed from the crawler's inability to interact with certain phishing pages, and possible client-server evasions leading to blank screens, or "404 Not Found" errors. In contrast, our crawling approach exhibited a significantly lower exclusion rate of 16.3%. This reduction reflects our system's improved resilience to evasion techniques and greater ability

to interact with dynamic phishing content. The remaining exclusions in our pipeline primarily involved static phishing pages that did not execute any JavaScript code, which is a prerequisite for our behavior-based analysis. To further quantify the improvement, we examined the 178 samples excluded by [Liu et al. 2024] crawler and found that our approach successfully processed 152 of them. This result implies that our method could reduce the discard rate from 43.5% to just 6%, demonstrating a substantial gain in sample coverage and thus increasing the robustness of phishing detection efforts in real-world conditions.

4.7. Baseline: Clustering Performance

We also evaluated our clustering performance against the most relevant prior work that applied clustering to execution traces, as proposed by [Sanchez-Rola et al. 2023]. Their approach applied a strict criterion for sample inclusion, ultimately limiting their approach to consider only 15 out of 409 available samples in their clustering process. While this method achieved a perfect Normalized Mutual Information (NMI) score of 1.0, the extremely limited sample coverage restricts its practical applicability. In contrast, our approach achieved an NMI score of 0.846—slightly lower in absolute terms—but was able to include 342 out of 409 samples, representing a 2.180% increase in sample coverage. This result underscores the broader generalizability of our method, which enables meaningful clustering performance while accommodating a significantly larger portion of the dataset.

5. Conclusion

In summary, we present a novel data collection methodology designed to capture evasive phishing samples—specifically, those that execute JavaScript instructions to fingerprint users before redirecting them to benign pages. Our approach significantly reduces the discard rate of phishing samples compared to existing classification tools, lowering it from 43.5% to just 6%. Additionally, we developed a clustering algorithm to evaluate the extracted features, achieving a Normalized Mutual Information (NMI) score of 0.846 while incorporating 342 out of 409 samples. In contrast, previous clustering efforts based on browser traces could only include 15 samples. Furthermore, we demonstrated the scalability of our instrumented browser by collecting 432,237 unique phishing samples over an eight-month period, and showed how our clustering pipeline can reveal long-term trends in phishing kit deployment.

Future work can build on the publicly available data collection and processing pipeline introduced in this study. Enhancing the clustering algorithm may yield even more accurate insights into phishing kit reuse and evolution. Moreover, extending the capabilities of the instrumented browser to capture additional data—such as rendering behavior or network request pipelines—could enable the analysis of static phishing pages that do not rely on JavaScript. The clustering algorithm could also be refined to detect shared Instruction Blocks across samples, potentially identifying different versions of the same phishing kit. With these improvements, the system could be deployed over longer periods to monitor the lifecycle of phishing kits, detect the emergence of new variants, and analyze the decline in usage of outdated kits.

6. Acknowledgments

This work was supported in part by CAPES, CNPq, ITA's Programa de Pós-graduação em Aplicações Operacionais (ITA/PPGAO), and by the São Paulo Research Foundation (FAPESP) grants #2020/09850-0, and #2022/00741-0.

References

Acharya, B. and Vadrevu, P. (2021). {PhishPrint}: Evading phishing detection crawlers by prior profiling. In *30th USENIX Security Symposium (USENIX Security 21)*.

APWG (2024). Anti-phishing working group trends report. `https://docs.apwg.org/reports/apwg_trends_report_q4_2024.pdf`. [Accessed in Apr 12, 2025].

Bijmans, H., Booij, T., Schwedersky, A., Nedgabat, A., and van Wegberg, R. (2021). 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.

Bitaab, M., Cho, H., Oest, A., Zhang, P., Sun, Z., Pourmohamad, R., Kim, D., Bao, T., Wang, R., Shoshitaishvili, Y., et al. (2020). Scam pandemic: How attackers exploit public fear through phishing. In *2020 APWG Symposium on Electronic Crime Research (eCrime)*.

Dambra, S., Sanchez-Rola, I., Bilge, L., and Balzarotti, D. (2022). When sally met trackers: Web tracking from the users' perspective. In *31st USENIX Security Symposium (USENIX Security 22)*.

Hao, S., Thomas, M., Paxson, V., Feamster, N., Kreibich, C., Grier, C., and Hollenbeck, S. (2013). Understanding the domain registration behavior of spammers. In *Proceedings of the 2013 conference on Internet measurement conference*.

Jueckstock, J. and Kapravelos, A. (2019). Visiblev8: In-browser monitoring of javascript in the wild. In *Proceedings of the Internet Measurement Conference*.

Kitphishr. Tool developed to discover available phishing kits hosted by the malicious url. `https://github.com/cybercdh/kitphishr`. [Accessed in Apr 12, 2025].

Kondracki, B., Azad, B. A., Starov, O., and Nikiforakis, N. (2021). Catching transparent phish: Analyzing and detecting mitm phishing toolkits. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 36–50.

Lee, J., Ye, P., Liu, R., Divakaran, D. M., and Chan, M. C. (2020). Building robust phishing detection system: an empirical analysis. *NDSS MADWeb*.

Li, B., Vadrevu, P., Lee, K. H., Perdisci, R., Liu, J., Rahbarinia, B., Li, K., and Antonakakis, M. (2018). Jsgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser javascript executions. In *NDSS*.

Liu, R., Lin, Y., Teoh, X., Liu, G., Huang, Z., and Dong, J. S. (2024). Less defined knowledge and more true alarms: Reference-based phishing detection without a predefined reference list. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 523–540.

Liu, R., Lin, Y., Yang, X., Ng, S. H., Divakaran, D. M., and Dong, J. S. (2022). Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. In *31st USENIX Security Symposium (USENIX Security 22)*.

MSDR (2024). Microsoft digital defense report. `https://www.microsoft.com/en-us/security/security-insider/intelligence-reports/microsoft-digital-defense-report-2024`. [Accessed in Apr 12, 2025].

Oest, A., Safaei, Y., Doupé, A., Ahn, G.-J., Wardman, B., and Tyers, K. (2019). Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists. In *2019 IEEE Symposium on Security and Privacy (SP)*.

Oest, A., Safaei, Y., Zhang, P., Wardman, B., Tyers, K., Shoshitaishvili, Y., and Doupé, A. (2020a). {PhishTime}: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *29th USENIX Security Symposium (USENIX Security 20)*.

Oest, A., Safei, Y., Doupé, A., Ahn, G.-J., Wardman, B., and Warner, G. (2018). Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*.

Oest, A., Zhang, P., Wardman, B., Nunes, E., Burgis, J., Zand, A., Thomas, K., Doupé, A., and Ahn, G.-J. (2020b). Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*.

Roesner, F., Kohno, T., and Wetherall, D. (2012). Detecting and defending against {Third-Party} tracking on the web. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*.

Sakurai, Y., Watanabe, T., Okuda, T., Akiyama, M., and Mori, T. (2020). Discovering httpsified phishing websites using the tls certificates footprints. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 522–531. IEEE.

Sanchez-Rola, I., Bilge, L., Balzarotti, D., Buescher, A., and Efstathopoulos, P. (2023). Rods with laser beams: understanding browser fingerprinting on phishing pages. In *32nd USENIX Security Symposium (USENIX Security 23)*.

Thomas, K., Li, F., Zand, A., Barrett, J., Ranieri, J., Invernizzi, L., Markov, Y., Comanescu, O., Eranti, V., Moscicki, A., et al. (2017). Data breaches, phishing, or malware? understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*.

Tian, K., Jan, S. T., Hu, H., Yao, D., and Wang, G. (2018). Needle in a haystack: Tracking down elite phishing domains in the wild. In *Proceedings of the Internet Measurement Conference 2018*.

Xu, W., Zhang, F., and Zhu, S. (2013). Jstill: mostly static detection of obfuscated malicious javascript code. In *Proceedings of the third ACM conference on Data and application security and privacy*.

Zhang, P., Oest, A., Cho, H., Sun, Z., Johnson, R., Wardman, B., Sarker, S., Kapravelos, A., Bao, T., Wang, R., et al. (2021). Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing. In *2021 IEEE Symposium on Security and Privacy (SP)*.