Automated Generation of Behavioral Signatures for Malicious Web Campaigns

Shaown Sarker[1][0009−0000−6700−5824], William Melicher[2][0000−0002−2505−684X], Oleksii Starov[2][0000−0002−2796−6345], Anupam Das[1][0000−0002−8961−9963], and Alexandros Kapravelos[1][0000−0002−8839−8521]

[1] North Carolina State University, Raleigh NC 27695, USA
{ssarker,anupam.das,akaprav}@ncsu.edu
[2] Palo Alto Networks, Santa Clara CA 95054, USA
{bmelicher,ostarov}@paloaltonetworks.com

Abstract. Web-based malicious campaigns target internet users across multiple domains to launch various forms of attacks. Extant research exploring the detection of such malicious campaigns involves applying supervised or unsupervised learning techniques on targeted campaign data producing machine learning models that are often expensive to train and are sluggish to react to the ephemeral nature of malicious campaigns. In this paper, we present an automated web-based malicious campaign detection system that produces campaign signatures representing both their static and dynamic behavior. We generated 379 campaign signatures that matched 36,427 unique malicious URLs with an extremely low false-positive rate (0.008%). We further applied our signatures on real world user traffic and identified 471 URLs, which were verified through VirusTotal and manual inspection. Our results provide valuable insight into web-based malicious campaign detection and our system could be utilized to improve existing defenses and the relevant field of threat intelligence.

1 Introduction

As the internet grows, more users than ever rely on it to perform various personal and professional activities such as communicating over social media, carrying out financial tasks, consuming entertainment, and fulfilling professional responsibilities. Unfortunately, malicious actors have evolved to target innocent victims on the web in a wide range of malicious activities, including promoting scams [14, 21, 30], coaxing users to click on malicious ads [35, 28], eliciting their credentials by faking a legitimate website [25], or stealthily stealing their clicks [2]. Often these attacks are carried out at scale on multiple domains to increase their effectiveness, resulting in a *web-based malicious campaign*.

Defenses against such campaigns often come in the form of blocklist services. Blocklists mark a URL and/or domain for malicious activity, causing the adversary to simply move the campaign content to a new unmarked URL/domain. Thus, creating a cat and mouse game between blocklists and the malicious

campaigner. Existing work in identifying campaigns involve various supervised and unsupervised learning or observation based on features extracted from targeted data associated with the specific campaign [14, 21, 30, 25, 35, 32, 31]. The resulting detection systems are often limited by their focus on specific type of attacks, and/or elaborate machine learning models that are sluggish to react to the short-lived nature of these campaign URLs.

We center our work in this paper on the observation that most malicious campaigns on the web share either static or dynamic behavior [25, 32, 31, 35]. In fact, recent malicious campaign detection systems rely on repeated behavior to cluster the campaign URLs and find malicious campaigns on the web [32, 35] using unsupervised learning. However, such learning models are often limited to only identifying campaign URLs that belong to the targeted campaign(s), and not campaigns of different types. Furthermore the majority of the web tends to skew towards benign content and fewer malicious URLs, making these models often subject to stringent low false-positive requirements, which a lot of them fail to achieve when applied to unlabeled data.

In this paper, we present an automated web-based malicious campaign detection system that identifies any campaign through signatures generated from repeating static and dynamic *behaviors* on URLs belonging to the campaign with very low false-positive rate. The signatures are robust against evasive maneuvers such as encryption and obfuscation, since they include dynamic behavior patterns along with static ones. Because of their simple construction and structure, along with the fast generation process, the generated signatures are quick to react for further detection of campaign URLs.

In our work, we crawled 2.8 million labeled URLs and generated 379 campaign signatures using our proposed approach. We were able to identify 36,427 malicious URLs from our labeled data belonging to malicious campaigns. We further applied our signatures on 431 thousand unlabeled URLs from real world user traffic and detected 471 unlabeled URLs belonging to 34 campaigns, which we confirmed to be malicious through VirusTotal and manual inspection. In summary, the contribution of this paper is as follows:

- We present an automated system for large-scale detection of web-based malicious campaigns through signatures of repeating behavioral patterns. Our system aims to be generic and not focus on any particular type of campaign.
- We perform deep instrumentation of the Chromium browser for tracking dynamic behaviors with significant detail. We use this custom browser to crawl our labeled URLs, and generate campaign signatures from the collected data. Our plan is to eventually make our signature generation system available to the research community (either by open-sourcing it, or by making it available as a service).
- We demonstrate that our generated signatures can successfully identify malicious URLs belonging to campaigns by evaluating them on both labeled and unlabeled URLs through oracles like VirusTotal and manual vetting.

– We compare our system against a production-ready deep learning classifier system and demonstrate that our system can complement such systems by identifying campaign URLs that are not detected by such systems.

2 Background

2.1 Malicious Campaigns

Malicious actors on the web replicate their attacks on multiple URLs on multiple domains to improve efficacy and scale of the attack. Web-based malicious campaigns are coordinated attacks that display the same malicious behaviors across URLs from multiple domains. These malicious behaviors are experienced by the end-user in the forms of network transactions for a resource (script, stylesheet, DOM content etc.), simple elements of the webpage, or even a particular piece of JavaScript execution.

2.2 Behaviors & Predicates

In the context of this paper, we use the term *behavior* on a webpage as both static and dynamic characteristics of the webpage that can be generalized and identified over other webpages. We use the term *predicates* as the textual representatives for these behaviors that we want to identify on a webpage. For this paper, predicates are dictionaries of key-value pairs of length two that represent a behavior. The first key-value pair contains the type of the behavior. The second key-value pair contains the properties representing the behavior which is a variable-length tuple consisting of text and/or number, e.g., `{type: "html_url", properties: ("http://example.org")}`. We give more comprehensive details of the predicates used in our work in §4.

2.3 Signatures

In our work, a campaign *signature* is an unordered conjunction of predicates. To apply a signature for a *match*, we extract our predicates from the candidate webpage and attempt to find *all* the predicates in the signature in the set of extracted predicates. We recognize a successful match when all the predicates in the signature are found on the webpage; otherwise we declare it as a non-match (see Figure 1).

2.4 Synthesizing Signatures

To generate campaign signatures from given labeled data, we identify the repeating behaviors that are most discriminating between malicious (positive) data and benign (negative) data and subsequently we want to see on which URLs we observed these behaviors. We synthesize a signature as the *least conjunction* of behavior predicates that represents these URLs as the footprint of malicious

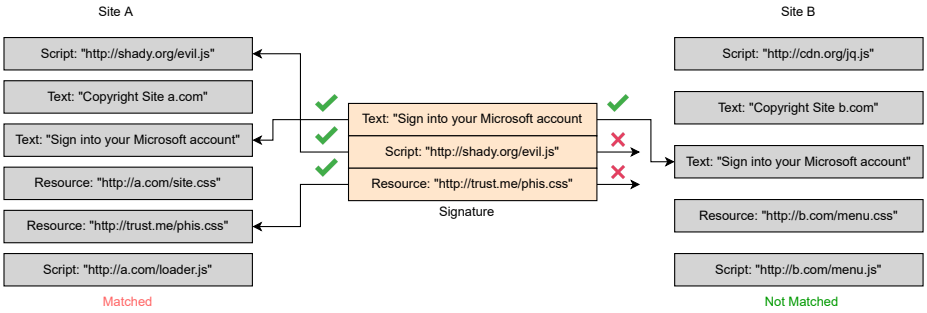

Fig. 1. Signature and matching process

campaign in our data. To achieve this, we take our inspiration from the domain of *learning logic programs through induction* [22, 10], specifically Relative least general generalization (*rlgg*) [26, 27]. The underlying settings for such learning directives can be broadly described in the following terms. Given a set of background knowledge B, along with a set of positive examples E^+ and a set of negative examples E^- each, we want to find a hypothesis consisting of predicates H such that the hypothesis in conjunction with the background knowledge explains the positive ground truths, $B \wedge H \vdash E^+$, and does not do so for the negative ground truth set, $B \wedge H \nvdash E^-$.

In our approach, we deviate from *rlgg* slightly by introducing our own simplified variation of partial ordering of predicates to generate the hypothesis (our signatures) that does not use substitution. However, similar to *rlgg*, we also forego the use of background knowledge and simply use ground truth examples to derive our hypothesis. We present our algorithm for synthesizing the signatures in full detail in §5.

3 Signature Structure

3.1 Limitations of Static Predicates

Static predicates are usually brittle and can become obsolete quickly, as the adversary is only required to minutely modify the static content of the page to evade signatures generated from static predicates solely. Furthermore, with the rise of techniques like HTML smuggling to generate the static content on the webpage dynamically [20], static predicate extraction is often hindered. To circumvent this, we complement static predicates with dynamic behavior predicates. Even if an adversary can take evasive measures to stem static predicate extraction or identification, predicates deduced from the dynamic execution of scripts on the webpage make the signatures robust enough to identify the malicious activity.

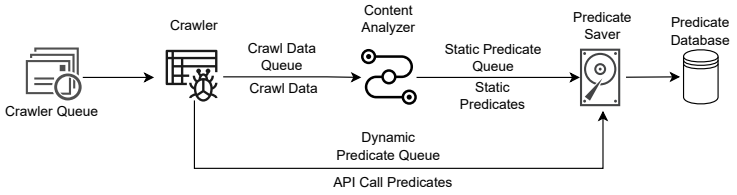

Fig. 2. Predicate extraction and collection from crawled webpages

3.2 Tool for Dynamic Predicates

Taking inspiration from prior research in the field of binary malware detection [10, 6, 15], we focused on browser API calls from script execution traces for our source of dynamic behavior predicates. Similar to system calls, browser API calls interact with the underlying system, in our case - the browser itself, to read or modify the state of it. Thus, like system calls, browser API calls can also contribute to the signature generation through dynamic behavior predicates representing the intent of JS-based malware. To get all browser API calls from dynamic execution, we leverage VisibleV8 [11], an open-source tool that logs all standardized browser API calls within the Chromium browser

VisibleV8 traces all invoked API calls along with JavaScript object property access or modification. For our work, we focused exclusively on browser API calls. However, VisibleV8 in its current state only gives the name of the browser API call being invoked and the script source code location where it was invoked from. To enhance the information gained from a browser API call, we wanted to include the parameter names and the corresponding argument values to the browser API call in our dynamic predicate. We extended the existing code-base of VisibleV8 by further instrumenting the V8 runtime library for our requirements. We built Chromium version 91.0.4472.101 with our instrumented V8. To counter API calls being invoked inside a loop, we extracted only unique API calls and argument tuples from a single webpage, without recording the same API call invocation with the same argument more than once.

4 Data Collection

4.1 Crawler

We designed a crawler based on the instrumented Chromium browser with our enhanced VisibleV8 variant (Figure 2). The crawler pulls a single URL from a queue and proceeds to visit the webpage of the URL using our instrumented browser. During each visit we collect all network requests made, the responses received along with the headers and bodies of all HTTP resources downloaded, all alert dialog types and their corresponding text (before being silently squashed for resuming page visit), and finally the rendered DOM-document along with the dynamic API call predicates. The static content is subsequently forwarded to a content analyzer and the API call predicates are stored via a predicate saver.

Predicate Type	Properties	Extracted from	Type
Traffic URL	Processed URL of originating request	Responses Received	Static
Traffic Domain	Host name of originating request	Responses Received	Static
Traffic IP	IP of the server	Responses Received	Static
Content Hash	SHA256 hash of response body	Responses Received	Static
Alert	Alert type, alert text	Intercepted Alerts	Static
HTML Attribute	Language and script attribute name, corresponding value of script tags	DOM Document Responses Received	Static
HTML URL	Tag name, processed URL extracted from specific attribute (see table 2)	DOM Document Responses Received	Static
HTML Domain	Tag name, host name of URL extracted from specific attribute (see table 2)	DOM Document Responses Received	Static
HTML Text	Processed text of length between 10 and 500 for HTML text tags except script and style	DOM Document Responses Received	Static
API Call	API name, [parameter name, corresponding argument value, ...]	VisibleV8 Log	Dynamic

Table 1. Extracted static and dynamic predicates during data collection

4.2 Content Analyzer

The content analyzer can extract nine types of static predicates from the collected static content. The predicates extracted can be divided into three categories:

Traffic Predicates. We iterate over the collected request-response pairs, and extract four distinct types of traffic predicates: the URL ❶ and the derived domain name ❷ from the URL of the request, and the IP address ❸ of the server that serves the response as predicates, and the SHA256 hash of the response body ❹. To generalize the traffic URL predicates, we breakdown the URL query string within the predicate and replace each parameter value with a positional placeholder value (`val1, val2, val3, ...`).

Alert Predicates. We extract the alert predicates ❺ as the combination of the dialog type and the dialog text before dismissing it during the visit.

HTML Predicates. We iterate over the responses received that have a status code of 2XX and a resource type of HTML, style, and script; along with the final DOM document to extract four predicate types. We extract the language and script attribute values for each script tag as attribute predicates ❻. For all textual HTML tags, we extract the text truncated to a preset bound (min 10, max 500) as text predicates ❼. We also perform filtering of random information within the text content such as phone number, and zip code, and replace them with generic placeholder value. For a predetermined set of HTML tags and their certain attribute values (see table 2), we extract the URL and the domain derived from it as URL ❽ and domain ❾ predicates, respectively. The extracted URL is similarly processed to have generalized query string parameters as in the case of traffic URL predicates.

Table 1 details all predicates, both static and dynamic, collected through our pipeline.

Table 2. HTML tags and corresponding attributes for extraction of HTML URL and Domain predicates

Tag Name	Attribute Name
script	src
a	href
form	action
img	src
object	data
iframe	src
frame	src
link	href

Table 3. Crawled URLs and collected predicates by verdict

URL Verdict	No. of URLS	No. of Predicates
Benign	1.7 million	1.2 billion
Malicious	784 thousand	183 million
Unlabeled	431 thousand	147 million
Total	**2.9 million**	**1.5 billion**

4.3 Collected Data

Using our data collection pipeline, we crawled and collected predicates from a variety of URL sources. For benign URLs, we crawled URLs from a sample of both Tranco [34] and Alexa [1] top 1-million sites. We crawled a portion of the VirusTotal [36] URL feed that have high-confidence detection (VT score > 3) for our malicious URLs.

To validate these signatures on a real user traffic, we make use of the URL filtering product from Palo Alto Networks. We crawled and analyzed unlabeled URLs from real world user traffic, which originated from web browsing, email links, etc., along with benign labeled URLs from the internet threat intelligence system from Palo Alto Networks.

Excluding URLs with empty content, we ended up with approximately 1.5 billion predicates from about 2.9 million URLs. Table 3 shows the breakdown of the crawled URLs and the collected predicates from them. Out of the ~1.5 billion predicates, dynamic predicates constitute 58.50%, the rest are static. The breakdown of collected predicates by type is shown in Table 4. The crawl of the URLs and predicate extraction were performed from the last quarter of 2021 to the first quarter of 2022.

Type	No. of Predicates
Traffic URL	95 million
Traffic Domain	21 million
Traffic IP	19 million
Content Hash	76 million
Alert	1 thousand
HTML Attribute	2 million
HTML URL	229 million
HTML Domain	37 million
HTML Text	162 million
API Call	907 million
Total	**1.5 billion**

Table 4. Collected predicates by type

5 Signature Generation

5.1 Ordering of Predicates

Our collected predicates belong to two overlapping sets - predicates from URLs with malicious verdict (positive set), and benign verdict (negative set). We want to order our collected predicates to determine the most prominent repeating predicates present in our positive set, but not included in the negative set. For each encounter of a predicate in the positive or negative set, we increase the corresponding count by one. We filter out all such predicates with a negative count of more than zero, and further discard predicates that do not have a certain positive count threshold. This is followed by ordering the remaining predicates by their positive count in descending order. We take the top predicates determined by a cutoff threshold, and construct the set of most discriminating repeated predicates. In essence, this process gives us the predicates that are most frequent on the malicious URLs, but are not observed at all on the benign URLs (see Algorithm 1). We present the process for tuning the values of the parameters used in our algorithm in §5.3.

5.2 Generating Signatures

For each predicate present in the constructed discriminative predicate set, we retrieve all malicious URLs with this predicate. We then retrieve the predicate sets for each of these URLs intersecting with the set of repeating discriminative predicate set. We then generate a single signature from the intersection of these predicate sets. The synthesized signature is essentially the minimal set of predicates that can identify the set of malicious URLs obtained in the prior step.

Next, we want to filter the generated signatures for two particular reasons. First, if we include signatures that can only identify a small number of webpages, then the signature is not very useful for identifying campaigns, and we would eventually end up with an inordinate number of signatures to apply. For this

Algorithm 1: Ordering predicates to construct set of repeating discriminative predicates

Data: P, set of predicates
F_{min}, minimum positive count threshold
C_{top}, top predicates count threshold
Result: D, set of discriminative predicates
$D \leftarrow \emptyset$;
for *each* $p \in P$ **do**
  $U_{pos} \leftarrow \{u \,|\, \text{predicate p is observed on positive URL u}\}$;
  $U_{neg} \leftarrow \{u \,|\, \text{predicate p is observed on negative URL u}\}$;
  if $|U_{pos}| <= F_{min}$ **then**
    continue;
  end
  if $|U_{neg}| > 0$ **then**
    continue;
  end
  $D \leftarrow D \cap p$
end
Sort D by $|U_{pos}|$ ascending;
$D \leftarrow \{p \,|\, \text{predicate p is in top } C_{top} \text{ of } D \text{ ordered}\}$

reason, we filter out any generated signatures that do not cover a minimum number of malicious URLs (the set of URLs from whose predicates the signature was generated).

Second, we want our generated signatures to have an certain balance. From a basic observation, a signature with a small number of predicates is more liberal and would match more webpages, whereas a signature with a large number of predicates would be hard to match and thus be conservative. We also discard signatures with a number of predicates below a certain threshold to avoid false positives. The signature generation and the subsequent filtering is displayed in Algorithm 2. We discuss these two filtering parameter optimization in §5.3.

5.3 Parameter Optimization

We have four parameter values in our signature generation algorithm we need to determine before applying the algorithm on real-world data: the parameters for determining the predicate order - the top discriminative repeating predicate count threshold C_{top}, and the minimum positive count threshold F_{min}; the parameters for filtering generated signatures - the minimum URL count threshold U_{min}, and the minimum predicate count threshold P_{min}.

Predicate count threshold C_{top}. We derived the distribution of our collected malicious predicate frequency and found that among the approximately 56 million malicious predicates, the frequency distribution is heavily skewed towards the last percentile. This implies that only $\sim$1% of these predicates show any form of repeating characteristics, and the rest of them are just randomly unique.

Algorithm 2: Generate signatures from the set of discriminative repeating predicates

```
Data: D, set of discriminate predicates
      U_min, minimum URL count threshold
      P_min, minimum predicate count threshold
Result: G, set of signatures generated
for each d ∈ D do
    U ← {u | predicate d is observed on URL u};
    /* Signature to be constructed                    */
    S_G ← ∅;
    /* URLs where predicate is observed               */
    O_U ← ∅;
    for each u ∈ U do
        P_u ← {p | predicate p is observed on URL u and p ∈ D};
        O_U ← O_U ∪ u;
        if S_G is ∅ then
            S_G ← P_u;
        else
            S_G ← S ∩ P_u;
        end
    end
    if |O_U| < U_min then
        continue;
    end
    if |S_G| < P_min then
        continue;
    end
    G ← G ∪ S_G;
end
```

Based on this observation, we selected the value of this parameter to 50,000 , which is a little smaller than ∼1% of 56 million.

Minimum positive count threshold F_{min}. From above, our discriminative repeating predicates were heavily skewed towards the single top percentile, with the highest positive frequency being 22,850 and the lowest being 4. For this parameter, we picked the value of 10 to ensure that all the predicates selected by the C_{top} value in the previous step is considered.

Minimum URL count threshold U_{min}. Inspired by hyperparameter optimization [9], We split our collected predicates into three distinct slices in the chronological order they were collected (see Table 5). We applied our algorithm with various values of this parameter while keeping all other parameters the same on the collected predicates till the 80-th percentile to generate signatures. We then apply the generated signatures on the next ten percentiles. The reason for this process is to avoid introducing bias in our signatures by generating signatures from data collected in the future to apply to past data. We keep the data

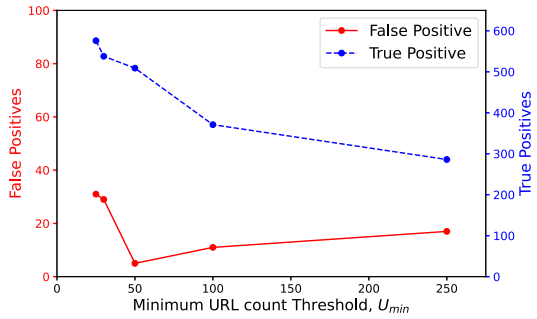

Fig. 3. Determining minimum URL count threshold parameter value

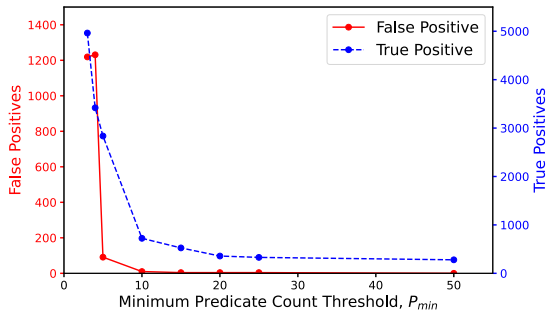

Fig. 4. Determining minimum predicate count threshold parameter value

from the last ten percentiles for our final evaluation as described in §6. Table 5 shows the labeled data slices used for each purpose in this paper.

To determine the performance of our selected parameter, we used the metric of keeping the false positive (FP) numbers as low as possible, while elevating the true positive (TP) numbers. Figure 3 displays how the various values of U_{min} affected the metric numbers, and based on this observation, we picked the value of this parameter to be 50.

Minimum predicate count threshold P_{min}. We followed the same strategy as in the prior paragraph to choose the optimal value for this parameter, as shown in Figure 4. From the observed values, we picked 5 for this parameter.

5.4 Generated Signatures

With the selected parameters from the prior section, we applied our algorithm on our collected predicates from the beginning to the 90-th percentile by the chronological extraction order of the predicates. During our predicates ordering phase, we processed 1.5 billion predicates, and consumed 50,000 predicates for

Table 5. Breakdown of labeled predicates used for each process by percentile slices (* in unlabeled evaluation, we applied the signatures generated from labeled data to unlabeled data)

Process	Generated From (percentiles)	Applied on (percentiles)
Parameter selection (see §5.3)	First to 80th	81st to 90th
Labeled evaluation (see §6.1)	First to 90th	91st to 100th
Labeled evaluation - regression (see §6.1)	First to 90th	First to 90th
Unlabeled evaluation (see §6.2)	First to 90th	Unlabeled data*

our signature generation phase (this number was bound by our selected parameters in the above section). This step using our setup took approximately 99 seconds. In our signature generation phase, we retrieved and consumed 14.5 million URL-predicate pairs from 309,721 malicious URLs, which was the biggest time consuming process in our case with 20 minutes and 32 seconds. After generating signatures and performing filtering, we ended up with 379 signatures from our data.

The signatures had 24,086 predicates in total, of which 21,067 (87%) were static, and 3,019 (13%) were dynamic API call predicates. The most specific signature consisted of 3,761 predicates, and the least specific one had 5 predicates (which was bound by our signature filtering parameter P_{min}, as discussed in §5.3), with an average of ∼64 and a median of 13 predicates per generated signature. Of the generated signatures, 183 (48%) had at least one dynamic predicate, and 196 (52%) consisted only of static predicates. There were 3 signatures that solely contained dynamic API call predicates.

6 Evaluating Generated Signatures

6.1 Labeled Data-based Evaluation

We applied the 379 generated signatures from §5, over the last ten percentiles of our labeled predicates (from 91th to 100th percentile by chronological order of extraction as mentioned in §5.3). In essence, this implies how signatures synthesized from previous data perform on detecting campaign URLs in unseen future data. We found 8,067 unique URL matches over 192 signatures in our last ten percentile data, with 41 labeled benign (from 41 matches) and 8,026 labeled malicious (from 8,028 matches). If we consider the number of labeled URLs in our application dataset, we have a false-positive rate of 0.008% and our generated campaign signatures covered 10.26% of the labeled malicious URLs that we were able to extract predicates from. This signature application process over this data took 1 minute and 53 seconds.

Table 6. Top 10 campaign signatures with highest toxicity

Rank	Signature Campaign Type	URL Matches
1	Fake domain sale scam	8,017
2	Phishing targeting Chinese visitors	2,823
3	Malware delivery	2,768
4	Fake domain sale scam	772
5	Facebook phishing	735
6	Phishing selling software solutions	502
7	Malware delivery	483
8	Phishing selling software solutions	431
9	Prize winning scam	416
10	Malware manipulating browsing history	332

We also applied our signatures on our collected labeled data itself in a regressive manner to evaluate coverage (see Table 5). While applying the signatures on the data used for generation (from first to 90th percentile of our labeled data) resulted in 28,401 unique malicious urls (from 28,470 matches) over 379 signatures, there were no false-positives. We sorted the signatures by their detection count to find the campaigns with the highest toxicity rate and manually examined the top 10 signatures to determine what kind of campaigns we were detecting and assigned them a campaign type, as shown in Table 6.

6.2 Detecting Campaign URLs in the Wild

As described in §4, along with labeled data, we also collected approximately 147 million predicates from about 431 thousand unlabeled URLs from a list of diverse sources (see §4.3). We wanted to see how our generated signatures perform when it comes to finding campaign URLs in the wild. After applying the 379 signatures, we found 483 matches from 34 signatures. This signature application process took 1 minute and 21 seconds.

We had 471 unique URLs from the matches of unlabeled data. We also performed a cross-check to find that none of the 471 URLs were matched from any of the 29 signatures that had at least one false-positive during our labeled data-based evaluation. To further verify our detection, we used VirusTotal [36], which provides detection results from an array of oracles. Out of the 471 URLs submitted to VirusTotal, we got 286 URLs (60.72%) that at least one VirusTotal oracle was able to identify having malicious activity.

Since a large portion of our detected campaigns is simply scams and phishing sites subject to block-listing, these are often short-lived and are either quickly taken down or moved to another random domain upon being flagged. Because of this, for the 185 URLs where VirusTotal did not report any malicious activity, we *manually* checked to confirm that all of the 185 URLs were either serving a scant benign page, or simply not alive. To determine what kind of malicious content they were serving during our crawling if any, we again resorted to manually

Table 7. Breakdown of manual analysis of URLs not flagged by VirusTotal

Campaign Category	URLS
Scam	124
Phishing	59
Malware Delivery	2
Total	**185**

inspecting the predicates of the signatures that matched them along with other predicates collected from these URLs. We were quickly able to identify that these URLs were indeed serving malicious contents involving various scams (credit card mining, lottery, dating site gathering personal info, fake survey, newsletter unsubscribe etc.), phishing (amazon card, Paypal business, Paypal pay in terms, domain hosting etc.), and delivering malware as displayed in Table 7.

To estimate the impact of our detection in the wild, we cross-checked our detected URLs against the enterprise customers' URL request logs for third quarter of the year from our partnering cybersecurity company. We found that 132 (28.03%) of our 471 unlabeled URLs detected through our signatures did show up in the customer logs. A total of 80,472 requests from 5,136 users were made to these 132 URLs, with an average of 609 requests per detected URL made, and an average of 16 requests made per user to these detected URLs. This demonstrates that the signatures for the campaigns were indeed active in the wild and targeting real world users.

6.3 Signature Case Studies

Case study: clickjacking campaign. Listing 1.1 displays a shortened version of our generated signature for clickjacking scams. In this particular clickjacking campaign, the user is duped into clicking on an element that is removed soon after, deceiving the user that the mouse click did not take place. We identified the signature match over 4 URLs with 3 different domains.

```
[
    // Redacted for brevity
    {
      "type": "api_call",
      "properties": [
        "Window.atob",
        "\"OTQ1NThOQVVTQOEyMTgzMDU3NzgwNzAwMDBDSA==\""
      ]
    },
    {
      "type": "api_call",
      "properties": [
        "Document.getElementById",
        "\"clickjack-button-wrapper-5\""
      ]
```

```
    },
    {
      "type": "api_call",
      "properties": ["Window.setTimeout", "\"clickjack_hider()\",5000"]
    }
]
```

Listing 1.1. Signature identifying clickjacking campaign

Case Study: history manipulation campaign. In this particular campaign, the webpage stuffs the users tab history while performing a number of redirects, and eventually landing on one of the random landing pages. When the user tries the back button on the browser, (s)he ends up visiting the corrupt URLs in her history pushed by the initial page. We matched this particular campaign signature across 332 URLs over multiple distinct domains. The identifying signature is displayed in shortened form in Listing 1.2

```
[
    // Redacted for brevity.
    {
      "type": "api_call",
      "properties": [
        "Element.setAttribute",
        "\"value\",\"http://dolohen.com/afu.php?zoneid=2468047\""
      ]
    },
    {
      "type": "html_url",
      "properties": [
        "form",
        "http://dolohen.com/?z=val1\u0026syncedCookie=val2"
      ]
    },
    {
      "type": "api_call",
      "properties": [
        "History.pushState",
        "#N,\"Redirect\",\"/afu.php?zoneid=2468047\u0026var=2468047\
    ↪ u0026rid=3V3cJ5LEtuPAKYxz6tD_Kw%3D%3D\""
      ]
    },
    {
      "type": "traffic_url",
      "properties": ["http://dolohen.com/afu.php?zoneid=val1"]
    },
    {
      "type": "traffic_url",
      "properties": ["http://dolohen.com/?z=val1\u0026syncedCookie=val2"]
    },
    { "type": "html_domain", "properties": ["form", "dolohen.com"] }
```

```
33 ]
```

Listing 1.2. Signature identifying JS malware manipulating browsing history

7 Contrast with ML-based Model

To the best of our knowledge, there is no existing ML model based system that attempts to detect web campaigns without targeting specific type of attacks like us. Although this obviates a direct comparison against a ML model based detection system, in this section we compare our system with a production-ready JavaScript malware detection framework named Innocent Until Proven Guilty (IUPG) [17].

IUPG is a static JS analysis framework that uses prototype learning to train deep neural networks for classifying URLs with JavaScript malware. Our version of IUPG was trained using the following dataset. For malicious data, a set of labeled high-confidence scripts from VirusTotal [36] containing 1.5 million scripts was used. Furthermore, similar scripts that VirusTotal tagged to belong to the same web campaigns (same vendor and VirusTotal tags) were sampled to 10% or at most 100 scripts to avoid over-fitting the model. For benign data, scripts extracted from crawling webpages from the Tranco [34] top URL list for a week were used. The trained IUPG was used to classify the same URL feed we used for our unlabeled data source as described in §4.3.

Upon cross referencing the unlabeled URLs detected using our generated signatures from the same unlabeled corporate client dataset (see 4.3), we found that out of the 471 URLs that were detected by our signatures, only 2 URLs were detected by IUPG. However, IUPG only classified URLs for malicious JavaScript (either inline, embedded, or fetched). For contrast, on 403 of these 471 URLs we were able to extract API call predicates from executed scripts. While IUPG provides high coverage as reported in [17], we show that our signatures can provide additional detection. Thus in a co-operative ensemble detection framework, our system can complement other systems like IUPG to improve detection rate significantly, and models like IUPG may incorporate similar behavioral features.

8 Discussion & Future Work

Reacting to campaigns. Our signature generation process takes approximately 22 minutes in total to generate signatures from 1.5 billion predicates. The involved and careful training process required for ML systems, such as IUPG (§7) is considerably slower. Retraining real-world ML-models can be expensive and only be done over a certain period of time. This significantly limits such ML-models to react to the constant shifting nature of web campaigns in the wild. Furthermore, we can easily generate signatures periodically to extend our signatures from newly available data - making our system react quicker to the campaigns. This makes our generated signatures more reactive to cyber threats compared to traditional IOCs such as hashes, domains, URLs, registry keys, etc.

Making the predicates more abstract. Abstract predicates contribute towards robust signatures. The more distinct the predicates are, the more likely they contribute towards conservative signatures. We transformed certain HTML text, HTML URL, and traffic URL predicates to be more abstract (§4.3). This can be further extended by introducing a predicate matching mechanism based on Levenshtein distance metric [38] for certain predicates, which should increase the coverage of the generated signatures over the missing known positive malicious URLs. However, we defer this as a potential future extension of our work as such signature generation requires its own study.

9 Related Work

There exist research works on scam campaign detection that focus on survey scams, tech-support scams, and fraudulent scams on free live-streaming services [14, 21, 28, 30]. SpiderWeb [33] constructs HTTP redirection graphs and extracts features of multiple categories to feed into a Support Vector Machine (SVM) classifier. Similarly, Surf [19] uses a J48 classifier on extracted features from poisoned search engine redirection graphs for detecting these URLs. WarningBird [18] is based on a SVM classifier on features extracted from URLs extracted from Twitter feeds. These works rely heavily on features extracted from behaviors that are used for training the supervised classifier. Our work in comparison is generating signatures using both static and dynamic behavior features that can react faster to malicious web campaigns.

Numerous research work has explored JS-based malware identification focusing on specific behaviors such as drive-by downloads [4, 23, 24], evasive JS malware [13, 16], obfuscated JS malware [12, 37, 8, 7], and other in-browser JS malware detection systems [5, 29, 6]. However, these research works focus on detecting JS malware rather than their campaign aspect. A handful of research has looked into detecting campaigns at scale and the similarity of web-based malware behavior. Prophiler [3] proposed a supervised fast filtering system for malicious webpage categorization. Starov et al. used static behavioral analysis to detect malicious campaigns in obfuscated JS [31]. Vadrevu et al. detected social engineering attack campaigns in low-tier ad networks [35]. Starov et al. identified malicious web campaigns through shared web analytic IDs among malicious websites [32]. Compared to these, we are proposing an automated system to generate signatures for malicious web campaigns of many types that does not require any highly targeted or correlated data for training.

10 Conclusion

In this paper, we propose an automated system that generates signatures for identifying malicious web campaigns at scale. We demonstrated that our system can handle large volume of data, and generate signatures quickly, making it reactive to the ever-changing campaigns. Our system can be used on its own

to trace down domains propagating malicious campaign contents to enrich existing blocking list services for threat intelligence. Alternatively, our system can complement other detection systems for achieving higher detection rates. From either perspective, our work provides valuable insight into the constantly morphing ecosystem of malicious web campaigns.

Acknowledgement

We thank the anonymous reviewers for their helpful feedback. This work was supported by the National Science Foundation (NSF) under grants CNS-2138138 and CNS-2047260. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

1. alexa.com. Alexa Top Sites. https://www.alexa.com/topsites. Accessed: 11-28-2021.
2. Stefano Calzavara, Sebastian Roth, Alvise Rabitti, Michael Backes, and Ben Stock. A tale of two headers: A formal analysis of inconsistent click-jacking protection on the web. In *Proceedings of the USENIX Security Symposium*, 2020.
3. Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler : A Fast Filter for the Large-Scale Detection of Malicious Web Pages Categories and Subject Descriptors. In *Proceedings of the International World Wide Web Conference (WWW)*, 2011.
4. Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *Proceedings of the International World Wide Web Conference (WWW)*, 2010.
5. Charlie Curtsinger, Benjamin Livshits, Benjamin G Zorn, and Christian Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *Proceedings of the USENIX Security Symposium*, 2011.
6. Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Song. Dynamic spyware analysis. In *Proceedings of the USENIX Annual Technical Conference*, 2007.
7. Aurore Fass, Michael Backes, and Ben Stock. HideNoSeek: Camouflaging Malicious JavaScript in Benign ASTs. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.
8. Aurore Fass, Robert P Krawczyk, Michael Backes, and Ben Stock. Jast: Fully syntactic detection of malicious (obfuscated) javascript. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2018.
9. Matthias Feurer and Frank Hutter. *Hyperparameter optimization*. Springer, Cham, 2019.
10. Matt Fredrikson, Somesh Jha, Mihai Christodorescu, Reiner Sailer, and Xifeng Yan. Synthesizing near-optimal malware specifications from suspicious behaviors. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2010.
11. Jordan Jueckstock and Alexandros Kapravelos. Visiblev8: In-browser monitoring of javascript in the wild. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC)*, 2019.

12. Scott Kaplan, Benjamin Livshits, Benjamin Zorn, Christian Siefert, and Charlie Curtsinger. " NOFUS: Automatically Detecting"+ String. fromCharCode (32)+" ObFuSCateD". toLowerCase ()+" JavaScript Code. In *Technical report, Technical Report MSR-TR 2011–57, Microsoft Research*, 2011.
13. Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *Proceedings of the USENIX Security Symposium*, 2013.
14. Amin Kharraz, William Robertson, and Engin Kirda. Surveylance: automatically detecting online survey scams. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2018.
15. Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *Proceedings of the USENIX Security Symposium*, 2009.
16. Clemens Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: De-cloaking internet malware. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.
17. Brody Kutt, William Hewlett, Oleksii Starov, and Yuchen Zhou. Innocent until proven guilty (iupg): Building deep learning models with embedded robustness to out-of-distribution content. In *2021 IEEE Security and Privacy Workshops (SPW)*, 2021.
18. Sangho Lee and Jong Kim. Warningbird: A near real-time detection system for suspicious urls in twitter stream. *IEEE transactions on dependable and secure computing*, 2013.
19. Long Lu, Roberto Perdisci, and Wenke Lee. Surf: detecting and measuring search poisoning. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011.
20. microsoft.com/. HTML smuggling surges: Highly evasive loader technique increasingly used in banking malware, targeted attacks. https://www.microsoft.com/security/blog/2021/11/11/html-smuggling-surges-highly-evasive-loader-technique-increasingly-used-in-banking-malware-targeted-attacks/. Accessed: 11-28-2021.
21. Najmeh Miramirkhani, Oleksii Starov, and Nick Nikiforakis. Dial one for scam: A large-scale analysis of technical support scams. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2017.
22. Stephen Muggleton, Cao Feng, et al. *Efficient induction of logic programs*. Citeseer, 1990.
23. Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. {WebWitness}: Investigating, categorizing, and mitigating malware download paths. In *Proceedings of the USENIX Security Symposium*, 2015.
24. Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Towards measuring and mitigating social engineering software download attacks. In *Proceedings of the USENIX Security Symposium*, 2016.
25. Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *Proceedings of the USENIX Security Symposium*, 2020.
26. Gordon Plotkin. Automatic methods of inductive inference. *Ph.D. Thesis*, 1972.
27. Gordon D Plotkin. A further note on inductive generalization. *Machine intelligence*, 1971.

28. M Zubair Rafique, Tom Van Goethem, Wouter Joosen, Christophe Huygens, and Nick Nikiforakis. It's free for a reason: Exploring the ecosystem of free live streaming services. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2016.
29. Paruj Ratanaworabhan, V Benjamin Livshits, and Benjamin G Zorn. Nozzle: A defense against heap-spraying code injection attacks. In *Proceedings of the USENIX Security Symposium*, 2009.
30. Bharat Srinivasan, Athanasios Kountouras, Najmeh Miramirkhani, Monjur Alam, Nick Nikiforakis, Manos Antonakakis, and Mustaque Ahamad. Exposing search and advertisement abuse tactics and infrastructure of technical support scammers. In *Proceedings of the International World Wide Web Conference (WWW)*, 2018.
31. Oleksii Starov, Yuchen Zhou, and Jun Wang. Detecting malicious campaigns in obfuscated javascript with scalable behavioral analysis. In *2019 IEEE Security and Privacy Workshops (SPW)*, 2019.
32. Oleksii Starov, Yuchen Zhou, Xiao Zhang, Najmeh Miramirkhani, and Nick Nikiforakis. Betrayed by your dashboard: Discovering malicious campaigns via web analytics. In *Proceedings of the International World Wide Web Conference (WWW)*, 2018.
33. Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2013.
34. tranco-list.eu. Tranco - A Research-Oriented Top Sites Ranking Hardened Against Manipulation. https://tranco-list.eu/. Accessed: 11-28-2021.
35. Phani Vadrevu and Roberto Perdisci. What you see is not what you get: Discovering and tracking social engineering attack campaigns. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC)*, 2019.
36. virustotal.com. VirusTotal - Analyze suspicious files and URLs to detect types of malware, automatically share them with the security community. https://www.virustotal.com/gui/home/upload. Accessed: 11-28-2021.
37. Wei Xu, Fangfang Zhang, and Sencun Zhu. Jstill: Mostly static detection of obfuscated malicious javascript code. In *Proceedings of the third ACM conference on Data and application security and privacy - CODASPY*, 2013.
38. Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 2007.