

Detecting Malicious Websites From the Perspective of System Provenance Analysis

Peng Jiang[✉], Jifan Xiao, Ding Li[✉], Member, IEEE, Hongyi Yu, Yu Bai[✉], Yao Guo[✉], Member, IEEE, and Xiangqun Chen

Abstract—Malicious websites are considered one of the top threats to the modern Internet. Thus, it is critical to effectively detect malicious websites for the security of the Internet. Conventional technologies typically rely on URL blacklists, or static and dynamic code analysis, which are known to have limitations. In order to effectively detect malicious websites, in this paper, we study malicious websites from the perspective of system provenance analysis for the first time. We first conduct a systematic feature engineering study on thousands of benign and malicious websites from the perspective of system provenance data. In our study, we discover eight useful features for malicious website detection. Based on these eight features, we propose ProvWeb, a novel non-intrusive system provenance-based tool, for malicious website detection. In our evaluation, ProvWeb can achieve an F1 score of 93.7% ~ 99.7% for the four combinations of browsers and OSes (Windows Chrome, Windows Firefox, Linux Chrome, Linux Firefox). This result confirms that the features discovered in provenance graphs are effective in detecting malicious websites.

Index Terms—System audit, website security.

I. INTRODUCTION

WEB technologies are ubiquitous today. At the same time, malicious websites have also become one of the most serious threats to modern Internet users [1], [2]. Malicious websites are used as vessels for more sophisticated malware [3], portals to collect massive personal and financial information [4], miners to generate crypto-currencies for monetary benefits [5] and tools for other malicious tasks. Malicious websites may impersonate other benign websites, which could wreak havoc on the reputation of legal websites.

Detecting malicious websites in the early stages could reduce the harm to both customers and faithful companies. However, it is also particularly challenging to detect malicious websites.

Manuscript received 14 April 2022; revised 12 March 2023; accepted 21 March 2023. Date of publication 29 May 2023; date of current version 16 May 2024. This work was supported in part by the National Key Research and Development Program under Grant 2022YFB4501802, in part by the National Natural Science Foundation of China under Grants 62172009 and 62141208, in part by Huawei Research Fund, and in part by the CCF-Tencent Open Fund. (*Corresponding author: Ding Li*)

Peng Jiang, Jifan Xiao, Ding Li, Yao Guo, and Xiangqun Chen are with the Key Laboratory of High-Confidence Software Technologies (MOE) School of Computer Science, Peking University, Beijing 100871, China (e-mail: pengjiang_pku2020@stu.pku.edu.cn; 2101111523@pku.edu.cn; ding_li@pku.edu.cn; yaoguo@pku.edu.cn; cherry@pku.edu.cn).

Hongyi Yu and Yu Bai are with the Advanced Institute of Information Technology, Hangzhou, Zhejiang 311200, China (e-mail: hyyu@aiit.org.cn; ybai@aiit.org.cn).

Digital Object Identifier 10.1109/TDSC.2023.3277613

Existing techniques are either prone to sophisticated evasive methods or have limitations in deployment.

One method to counter malicious websites is using blacklists of malicious URLs [6]. However, adversaries may use new URLs or benign domains to dodge the detection of *blacklists*. As reported, 40% of malicious web pages are in good domains [7]. Detecting malicious websites in benign domains requires a very fine-grained blacklist, which is daunting to build.

Another popular detection technique is *static code analysis* [8], which detects malicious websites based on static code features of HTML and JavaScript. The limitation of static code analysis is that it is prone to code obfuscation and camouflaging [9].

Unlike static code analysis, *dynamic code analysis* seeks to monitor the runtime behaviors of browsers and JavaScript engines while visiting websites [10]. By having runtime information, dynamic analysis methods are more robust to code obfuscation and camouflaging. Hence, they are potentially more accurate than static analysis techniques. However, dynamic analysis techniques may still be vulnerable to camouflaging if they use some static code-level features [11]. Furthermore, existing dynamic code analysis techniques require either modifications to browsers or the building of plugins [10], [11], [12], which limits the scope of applications of these techniques.

To better detect malicious websites, it is useful to study the features of websites from a perspective different from conventional techniques. Recently, system provenance analysis [13], a non-intrusive technique that monitors system behaviors of processes passively, has been applied to detect several types of stealthy and complex attacks [14], [15], [16], [17]. Particularly, a few recent studies have shown that system provenance analysis can capture subtle system runtime features of processes [18], which then leads to file-less attack detection [19], malicious installation package detection [20], and abnormal process detection [21].

Inspired by previous work, we argue that *system provenance is a useful feature, which is orthogonal to existing features, for malicious website detection*. System provenance analysis is dynamic, so it is less prone to URL changing and code camouflaging. Unlike existing dynamic analysis, system provenance analysis does not require modifications to browsers or the development of add-ons. In summary, system provenance analysis has the potential to detect malicious websites effectively. We summarize the differences between provenance-based techniques and conventional techniques in Table I.

TABLE I
COMPARISONS BETWEEN DIFFERENT MALICIOUS WEBSITE DETECTION METHODS

Approaches	Category	Criteria		
		Robustness	Runtime-Cost	Deployment
Static Analysis	URL Blacklist [6], [7]	Low	Low	Easy
	Static Code Analysis [8], [9]	Low	High	Hard
Dynamic Analysis	Behavior Detection [33]	Low	Low	Easy
	Dynamic Code Analysis [10], [11], [12]	High	High	Hard
Provenance Analysis	System Monitor	High	Low	Easy

Robust means how Robust the approach can prevent camouflaging. Runtime-cost is the system overhead introduced by different approaches. Deployment is how easily we can deploy different approaches without modifications to browsers or building plug-ins.

Unfortunately, there is no systematic study on how system provenance analysis could be applied to detect malicious websites. To close the gap between system provenance analysis and malicious website detection, in this paper, we conduct the very first systematic study on detecting malicious websites from the perspective of system provenance analysis. The main research question of our study is: *can system provenance analysis techniques be used to detect malicious websites?*

To answer this question, we first conduct a systematic feature engineering study on realistic malicious websites. We compare the system provenance data of 4,000 malicious websites to their counterparts of 4,000 benign websites. We discover several important features that can be used to detect malicious websites. *Syntactically*, we find that the number of nodes, the number of edges, and the width of the provenance graph of a website can be used to detect some obviously benign websites. *Semantically*, we find that the distributions of file read, file write, dynamic library usage, IP address accessed, and operation types of malicious websites are also substantially different from their counterparts.

Based on our discovery, we propose the very first online provenance-based malicious website detection tool, ProvWeb, which abstracts the features we have discovered in our measurement study with a novel algorithm and then feeds the features to a well-designed classification model. Unlike conventional dynamic analysis techniques, ProvWeb is non-intrusive, which means it *DOES NOT* require any modifications to browsers. Being non-intrusive may broaden the application scope of ProvWeb. In our evaluation, ProvWeb can achieve an F1 score of 93.7% ~ 99.7% for the four combinations of browsers and OSes (Windows Chrome, Windows Firefox, Linux Chrome, Linux Firefox). Based on our study, we conclude that system provenance analysis is capable of detecting malicious websites.

We summarize the contributions of this paper as follows:

- To the best of our knowledge, our study is the very first large-scale systematic study on malicious website detection from the perspective of system provenance analysis.
- Our study confirms that system provenance data of malicious websites are different from their counterparts of benign websites. Notably, we discover three syntactic features and five semantic features that can be used to detect malicious websites.
- We propose the first system provenance analysis-based tool, ProvWeb, for malicious website detection. Unlike other dynamic analysis tools, ProvWeb does not require any modifications to browsers.

- We conduct a thorough evaluation on ProvWeb.
- We build a dataset containing the system provenance data of 4,000 benign websites and 4,000 malicious websites for four different configurations of OSes and browsers.¹

Other parts of this paper are organized as follows: in Section II, we discuss the background information and motivating examples; in Section III, we discuss the threat model and assumptions of our paper; in Section IV, we discuss the result of our empirical feature engineering study; in Section V, we discuss our design of ProvWeb; in Section VI, we discuss the evaluation of ProvWeb; in Section VII, we discuss related work; in Section VIII, we discuss possible threats to the validity of our approach; in Section IX, we make conclusions about this paper.

II. BACKGROUND AND MOTIVATION

In this section, we discuss the background information and the motivation that drives us to conduct this work.

A. Malicious Websites

Malicious websites are increasingly common today. URLs used in malicious webpages mainly contain three types in terms of their role [22]. Adversaries use malicious websites to deliver viruses [3], collect personal data [23], mine cryptocurrencies [5], and perform other malicious tasks. One common type of malicious website is *drive-by download* [3]. Adversaries build malicious web pages that, once visited, another malware will be inadvertently installed onto the victim's host. Drive-by downloads have caused multiple significant incidents recently. For example, the Lurk criminal group steals 45 million dollars with drive-by download and other techniques [3].

Another common type of malicious website is phishing websites [23], which impersonate legitimate websites to lure victims into leaking their critical information, such as credentials for bank accounts. Besides individual targets, adversaries may also use phishing websites to attack enterprises, which may cause a substantial amount of monetary loss. Although tremendous efforts have been taken to address phishing websites, their threats are still increasing. As reported by PhishTank [24], there are at least hundreds of new phishing websites discovered every day.

¹Our data is available at: <https://github.com/provweb/provenanceweb>

Cryptojacking [5], [25] websites, such as CoinHive [26], also gained popularity recently because of the rising of cryptocurrencies. Adversaries may embed scripts that mine cryptocurrencies, such as Monero [27], which is a website that looks normal and gains monetary benefits from mining. Cryptojacking websites can cause high CPU usage to the victim's computer, which may decrease the lifetime of the victim's device. Furthermore, adversaries may take advantage of the vulnerabilities of a benign website to embed cryptocurrency mining scripts, which may wreak havoc on the usability of the benign website.

Besides drive-by download, phishing, and cryptojacking, there are multiple novel types of malicious websites [28]. First, attackers inject scripts that specifically target authenticated site administrators. The notorious Magecart malware, which steals credit card details from checkout pages of eCommerce sites by running a variety of different malicious JavaScript, spreads through multiple online stores [29]. Second, websites also suffer supply-chain attacks. For example, attackers may build malicious scripts like widgets, tracking services, ad scripts, site counters, or social add-ons that may lead to malicious behavior on benign websites [30]. Third, attackers also build SEO spam to manipulate the results of search engines [31].

Unfortunately, detecting malicious websites is a challenging task. Because of the flexibility of HTML and JavaScript, adversaries can easily obfuscate their code to bypass detectors. In practice, detection techniques often suffer many problems, such as incomplete features, ineffective models, etc. [32].

B. System Provenance

System provenance represents the log that records the communications between applications and the OS. Typically, system provenance data include file operations, network operations, and process operations such as fork and IPC [13], [17]. System provenance data are often represented by a graph model [13]. We will show an instance of such a graph in the next section as a motivating example. System provenance data are traditionally used for hunting APT attacks [14], [17], [21], [34], [35]. However, recent studies also show that system provenance data can substantially expose the running behaviors of applications [18]. Based on this result, people have developed multiple techniques to detect hijacked processes [19] and stealthy malware [36].

C. Motivating Example

In our observation, malicious websites and benign ones are fundamentally different. These differences can leave hints in provenance graphs. Particularly, we find two types of features in provenance graphs of malicious websites, which are resulted from the fundamental differences in functionalities and design logic between malicious and benign websites. First, malicious websites may leave *positive features* in provenance graphs. Malicious websites may use certain system resources that are rarely used by benign websites since they need to fulfill certain malicious requirements that do not belong to benign websites. On the flip side, malicious websites may also leave *negative features* in provenance graphs. Due to different goals, a malicious website may also ignore some design requirements of a benign

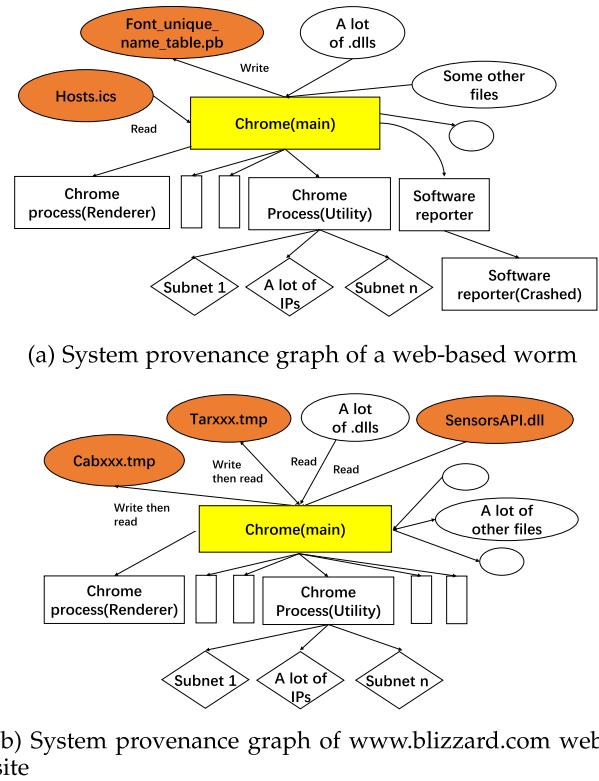


Fig. 1. A motivating example.

website, such as quality control. Thus, a malicious website may not have some important system resources of benign websites in its provenance graph.

We use two realistic websites in Fig. 1 as examples. Fig. 1(a) is the provenance graph of a website that contains the Ramnit worm. Fig. 1(b) shows the provenance graph of a benign website www.blizzard.com. Fig. 1(a) contains two positive features. First, it accesses the Host.ics file, which is used by Microsoft Internet Connection Sharing. While it is a piece of important information for worms to discover the next target in the same LAN, according to our study for 4000 benign websites, NONE of them use this file. Second, the virus in Fig. 1(a) writes some payloads to a file named "font_unique_name_table.pb" in the browser cache. This operation is also uncommon in benign websites.

Fig. 1(a) also contains two negative features. First, the benign website in Fig. 1(b) compresses its communicated data. The compression leaves temporary files, such as Cab.tmp and Tar.temp, in the provenance graph. Although data compression is a common practice in benign websites, we observe that few malicious websites compress their data. Similarly, we observe that the benign website in Fig. 1(b) uses SensorAPI.dll to read users' geological information to provide personalized services. However, in our study for 4000 benign websites and 4000 malicious websites, the probability for a malicious website to use SensorAPI.dll is a magnitude lower than benign websites. In other words, malicious websites rarely use sensor information.

In summary, the example in Fig. 1 indicates that the provenance graphs of malicious websites can be fundamentally different from benign websites. This observation motivates our

TABLE II
THE SYSTEM ENTITIES AND EVENTS WE CONSIDER

Subject	Object	Attributes	Events
Process	Process	Executable path, Pid, Host name	Fork, End
	File	File path, Host name	Read, Write, Execute, Delete
	Network	Src IP, Src port, Dst IP, Dst port	Read, Write, Connect, Disconnect, Accept

study to detect malicious websites based on features in system provenance graphs. In this paper, one of the key goals is to find *general features* that can separate malicious websites and benign websites.

III. PROBLEM DEFINITION AND THREAT MODEL

We aim to characterize features that may separate the running behaviors of malicious websites from benign websites. We define the system provenance graph of a website² as an attributed graph, $G = \langle N, E, l_n, r \rangle$. N is the set of nodes, which contains the system subjects and system objects. In this paper, system subjects and system objects are listed in Table II. E is the set of edges, which are system events made by system subjects to system objects. The types of events are also listed in Table II. l_n represents the label function of nodes, where $l_n(N)$ returns the attributes of a node, such as file names, process IDs, and executable names. $r \in N$ is the root node of the graph, which is the browser process that accesses the target website.

The problem we want to solve in this paper is: given a provenance graph of websites M , we aim to answer whether M is generated by a malicious website or a benign website.

We make typical assumptions of system provenance analysis [17], [20], [21]. We assume that the OS kernel is not compromised by adversaries and system provenance data is accurate. Our threat model also assumes the integrity of transmission and storage of provenance data.

IV. FEATURE ENGINEERING

The main goal of this paper is to answer whether provenance data is effective in detecting malicious websites. To achieve this goal, we first conduct an empirical study on thousands of realistic malicious websites. We aim to find effective features in provenance data that can separate malicious websites from benign websites. Particularly, we answer the following two sub-questions

- 1) Are there *syntactically* level features?
- 2) Are there *semantically* level features?

A. Datasets

We collect samples of malicious websites from two different sources: VirusShare [37] and PhishTank [38] where VirusShare is a collection of known malware samples, and PhishTank is a collection of up-to-date malicious websites URLs. For

²In our paper, a website is a synonym of the main page of the website

VirusShare, we crawl its samples that are in HTML format dated from 2019/8/10 to 2020/11/10. We remove those samples that are invalid, e.g., empty files. In the end, we obtained 1,500 valid VirusShare samples. For PhishTank, we monitor its latest updates every day from 2020/12/25 - 2021/1/10. We only select the URLs that are marked as verified and remove those that cannot be accessed. We have collected a total of 2500 samples from PhishTank. Thus, we have used 4,000 malicious samples in total in our experiments. To build benign samples, we randomly collected 4,000 samples from Alexa's top 10,000 websites [39].

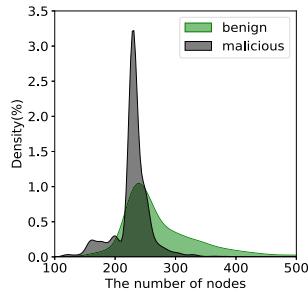
B. Experimental Environment and Data Collection

We build our experimental environment based on Cuckoo Sandbox [40]. We run Cuckoo on four desktops with an I7 CPU, 32 GB memory, and Ubuntu 18.04 OS. On each desktop, we run one Cuckoo instance. We select Chrome and Firefox as our target browsers. We chose them because they and their variants represent more than 81% market share in 2020.

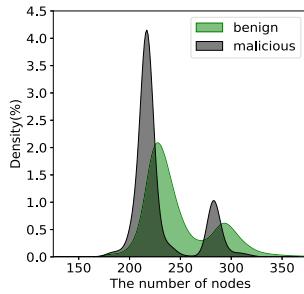
To collect system provenance data, we implement agents that monitor Windows ETW and Linux Sysdig events. Similar agents have been used in previous papers [16], [17], [20], [21], [41]. Types of system events collected by our agents are shown in Table II. Similar to related work [17], we collect system provenance data to a remote MongoDB. We implement the tool that generates provenance graphs based on Nodoze [17].

In order to simulate realistic scenarios, we upload our samples downloaded from VirusShare to an Apache server and access each sample remotely. We then deploy the agents to Cuckoo Sandbox and collect system provenance data. To generate provenance graphs for each website, we visit the URL of each target website for 45 seconds and collect all provenance data generated by browsers. To ensure the generated data is realistic, we randomly generate events such as clicking and scrolling under the protocol proposed in related work [42]. To ensure that Cuckoo does not undermine the validity of our study in realistic environments, we manually compared the provenance graph of 100 websites generated in both Cuckoo Sandbox and bare-metal machines. We find that the provenance graphs from Cuckoo Sandbox are identical to their counterparts in bare-metal machines, except for some events from Cuckoo's agent. We then filtered out the events generated by Cuckoo's agent to make sure our results are realistic.

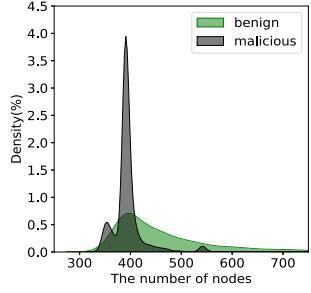
To ensure the soundness of feature engineering, we use both visualized and statistical methods. For syntactical features, we first plot the distributions of the number of nodes, the number of edges, and the widths of provenance graphs for benign and malicious websites. Then, we use the Two Independent Samples Non-parametric Test [43] to test the statistical significance of the differences between the two distributions. We chose the Two Independent Samples Non-parametric Test because it does not require the assumption of normal distribution for the data. In our statistical test, the null hypothesis is that the distributions of the number of nodes between benign and malicious websites are the same. We use p-value [44] as the evaluation metric and reject the null hypothesis when $p < 0.05$.



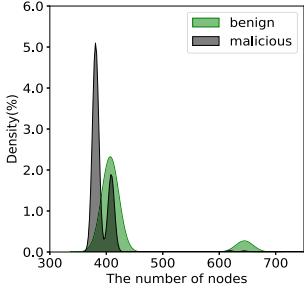
(a) Chrome, Windows



(b) Firefox, Windows



(c) Chrome, Linux



(d) Firefox, Linux

Fig. 2. The distribution of the numbers of nodes.

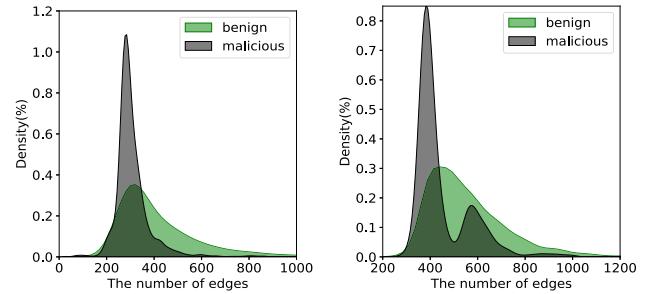
For semantic features, we first visualize the high-dimensional features with t-SNE [45]. Then, we build a logistic regression classifier for the Network feature and a k-NN classifier for other semantic features to statistically measure how well can the semantic features detect malicious websites. We run the two classifiers with ten randomly generated test datasets. We then report the detection accuracy. Finally, we use Kolmogorov-Smirnov [46] test to ensure that the detection accuracy is statistically higher than a random guess [47]. In our statistical test, we say the detection accuracy is statistically higher than a random guess if the p-value is smaller than 0.05.

C. Syntactic Features

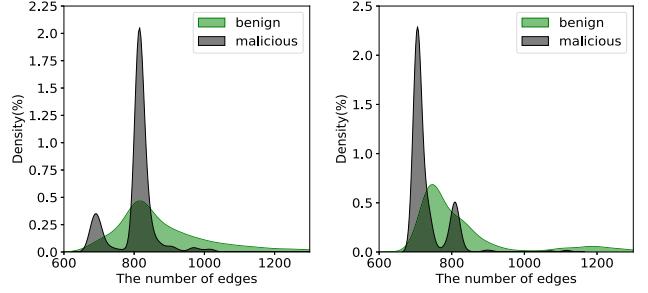
To study features of malicious websites on the syntactic level, we measure three metrics in our study: the number of nodes, the number of edges, and the width of provenance graphs. Particularly, we answer the following questions:

- RQ 1: Do benign and malicious websites have different numbers of nodes?
- RQ 2: Do benign and malicious websites have different numbers of edges?
- RQ 3: Are the widths of provenance graphs different between benign and malicious websites?

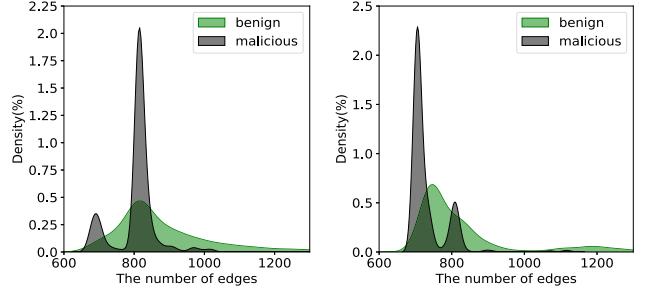
1) *Number of Nodes*: Fig. 2 shows the distribution of the number of nodes for all malicious and benign websites in different browsers in different OSes. Interestingly, we find that there exists a threshold that separates some benign websites from malicious ones. For instance, for Chrome on Windows, a website with more than 300 nodes in its provenance graph is 20 times more likely to be benign than malicious. However, for websites with fewer than 260 nodes, it is hard to separate



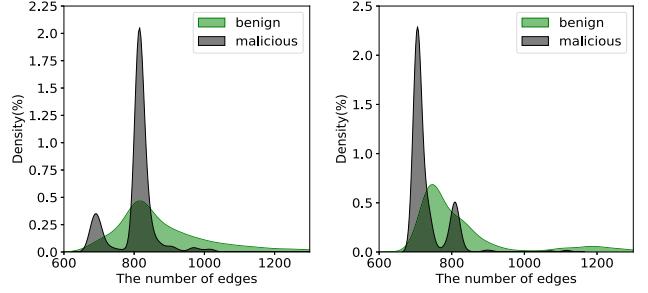
(a) Chrome, Windows



(b) Firefox, Windows



(c) Chrome, Linux



(d) Firefox, Linux

Fig. 3. The distribution of the numbers of edges.

malicious and benign websites purely based on the number of nodes. We also have similar observations on the other three combinations of browsers and OSes. Thus, we argue that the number of nodes in provenance graphs can help narrow down the range of benign websites and increase the accuracy of malicious website detection.

Besides visually comparing the distributions, we run the Two Independent Samples Non-parametric Test [43] to test the statistical significance of the differences between the two distributions. The result is in Table III, which shows that for all configurations, the p-values are smaller than 0.05. Thus, we reject the null hypothesis.

2) *Number of Edges*: Fig. 3 shows the distribution of the number of edges of all malicious and benign websites for different browsers in different OSes. We observe similar phenomena as in the case of the number of nodes: when the number of edges of a provenance graph is larger than a threshold, the provenance graph is not likely to be generated by a malicious website. In our study, for the four combinations of browsers and OSes, when the numbers of edges are larger than 500, 750, 880, and 840, respectively, the corresponding websites are 10 times more likely to be benign than malicious. Similarly, we conclude that the number of edges is also an effective feature to help detect certain websites that are obviously benign.

We also run the Two Independent Samples Non-parametric Test and report the p-values in Table III, which shows that the p-values for all configurations are smaller than 0.05.

3) *Widths*: To define the width of a provenance graph, we first define the temporal spanning tree of a provenance graph. Let TE be the set of edges in the temporal spanning tree of a provenance graph, $IN(n)$ be the set of in-edges

TABLE III
STATISTICAL DIFFERENCES BETWEEN MALICIOUS AND BENIGN WEBSITES FOR SYNTACTIC FEATURES

	Chrome, Windows			Firefox, Windows			Chrome, Linux			Firefox, Linux		
	Mean	Standard Deviation	p-value	Mean	Standard Deviation	p-value	Mean	Standard Deviation	p-value	Mean	Standard Deviation	p-value
Node	279/228	78/34	****	247/229	33/28	****	467/394	109/32	****	413/390	72/26	****
Edge	442/308	256/82	****	553/451	179/126	****	891/806	160/60	****	828/732	144/53	****
Width	223/197	52/30	****	242/225	33/28	****	312/321	20/16	*	339/305	24/15	****

For mean and standard deviation, the values of Benign websites and Malicious websites are divided by “/”. P values less than 0.05 and larger than 0.01 are summarized with one asterisk. P values less than 0.01 are summarized with two asterisks. P values less than 0.001 are summarized with three asterisks, and P values less than 0.0001 are summarized with four asterisks.

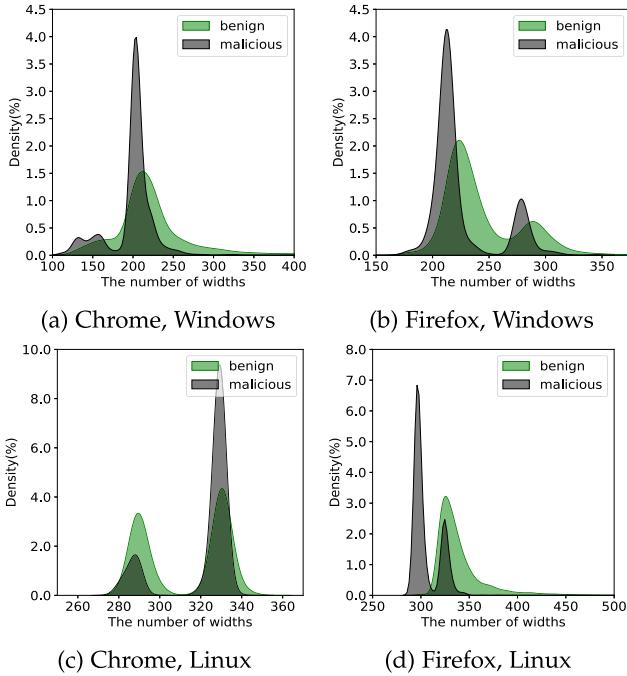


Fig. 4. The distribution of widths.

of node n , we have $e_1 \in IN(n) \rightarrow e_1 \in TE(n) \iff \forall e_2 \in IN(n), t(e_1) < t(e_2)$, where $t(e_1)$ represents the time stamps of edge e . A similar concept of temporal spanning tree is also used in previous work [20], [48]. Then the width of a provenance graph is defined as the maximum number of nodes on each layer of the temporal spanning tree of the provenance graph.

Fig. 4 shows the distribution of the widths of all websites. Similarly, we find that there is a threshold for the width that separates certain obviously benign websites from malicious ones. Take Fig. 4(a) as an example. A graph with a width of 215 is equally likely to be benign or malicious. However, when the width is 250 and more, the provenance graph is five times more likely from a benign website. In summary, we also conclude that the width is an effective feature to help detect obviously benign websites.

Table III shows the p-values of the Two Independent Samples Non-parametric Test. We conclude that the widths of provenance graphs of benign and malicious websites are statistically different.

4) *Conclusion on Syntactic Features:* Our study shows that although syntactic features cannot directly detect malicious

websites, they are effective in detecting those obviously benign websites and reduce the search space for other features. Specifically, we find a website that generates a “big” provenance graph with more nodes, more edges, and wider than a certain threshold are more likely to be benign. These features can help remove some obviously benign websites and improve the accuracy of detection. Based on our study, We believe that syntactic features can be useful in helping distinguish benign websites from malicious websites.

D. Semantic Features

To study the features of malicious websites on the semantic level, we study how differently malicious websites use files, networks, libraries, and other processes from benign websites. We follow the same protocol and use the same data set as Section IV-C. Particularly, we answer the following research questions:

- RQ 4: Do malicious websites use files differently from benign websites?
- RQ 5: Do malicious websites use dynamic libraries differently from benign websites?
- RQ 6: Do malicious websites and benign websites have different patterns for operation types?
- RQ 7: Do malicious websites and benign websites have different patterns for network operations?
- RQ 8: Do malicious websites and benign websites have different patterns for process operations?

1) *Files:* To answer if malicious websites use files differently from benign ones, we measure the distributions of file extensions accessed by malicious and benign websites. Specially, we build histograms for file read operations, and file write operations separately. The read histogram is a key-value map where keys are extensions of files read by a website, and values represent how many times each extension has been read. The definition of the write histogram is similar except that the operation is changed to file write. For instance, assuming a website reads A.dat, B.dat, and C.png and writes to D.dat, E.tmp, and F.tmp, we build two histograms for the site: the read histogram, which is $\{.dat : 2, .png : 1\}$ and the write histogram, which is $\{.dat : 1, .tmp : 2\}$. The read histogram and the write histogram are used as two separate feature vectors.

To visualize the differences between malicious websites and benign websites in read-histograms and write histograms, we plot them with t-SNE [45]. The visualization of read-histograms is shown in Fig. 5 for different browser and OS configurations.

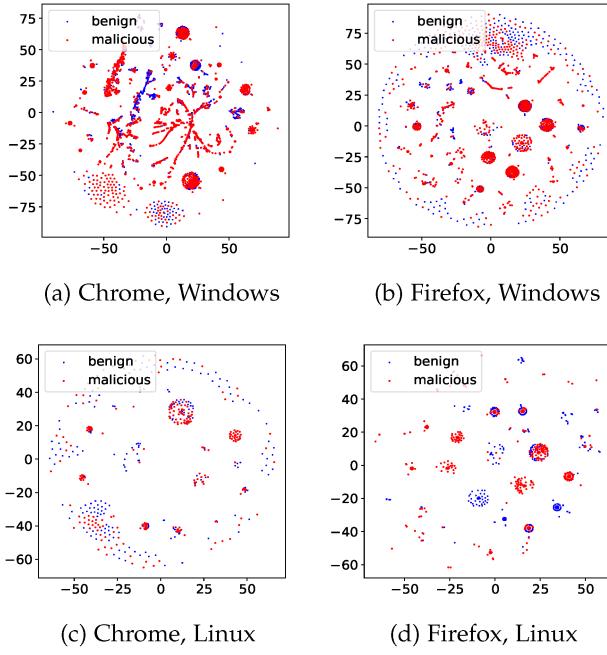


Fig. 5. Visualization of read histograms.

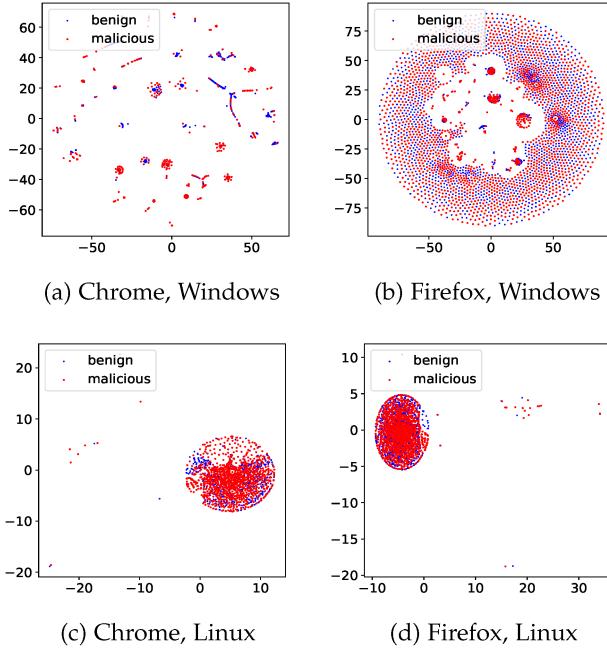


Fig. 6. Visualization of write histograms.

In the figure, red represents malicious samples while blue represents benign samples. Similarly, we plot the write histograms of websites with different configurations in Fig. 6.

We can see from Fig. 5 that the read histograms of malicious websites and benign websites are well clustered. For instance, in Fig. 5(a), there are a few clusters that are dominated by either malicious or benign websites. Fig. 5(c) and (d) also contain similar clusters. It seems that Fig. 5(b) does not have obvious clusters of either type of website, but we can still find

TABLE IV
THE ACCURACY OF kNN CLASSIFIER ON HISTOGRAM FEATURES

	Chrome Windows	Firefox Windows	Chrome Linux	Firefox Linux
File Read	70.8%	57.3%	71.2%	87.2%
File Write	66.4%	52.0%	54.6%	53.0%
Operation	82.7%	89.7%	81.9%	91.2%
Library	64.8%	51.0%	66.4%	68.3%

P-values are all smaller than 0.05.

a hyperplane (e.g., SVM) that separates benign websites from malicious websites in higher-dimensional space.

Write histograms are less effective but still useful for malicious website detection. Only Fig. 6(a) contains obvious clusters. This is because the dimension of the writ histograms is only half of the read histograms, which means that websites write to fewer types of files than they may read. Nevertheless, write histograms can also help separate benign websites from malicious websites.

To statistically evaluate whether malicious websites use files differently from benign websites, besides visualization, we also use k-Nearest Neighbours (kNN) to evaluate how samples from the same class clustered quantitatively. kNN predicts the labels of a target sample based on a number of data points surrounding the target. If kNN achieves a good prediction accuracy in our data, it means samples from the same category are well clustered together. To evaluate our read and write histograms with kNN, we randomly choose 800 websites from our data. *Half* of them are malicious and *half* of them are benign. Then we use other samples in our data to predict the labels of the 800 websites. To ensure the soundness of our experiment, we repeat the experiment ten times with different randomly sampled test data. We also use the Kolmogorov-Smirnov [46] test to ensure that the accuracy is statistically higher than 50%, the accuracy of a random guess. We report the prediction accuracy and the corresponding p-value in Table IV.

Table IV shows the prediction accuracy for read histograms is from 57.3% to 87.2%. For write histograms, the accuracy is from 52.0% to 66.4%. The p-values are all smaller than 0.05, which indicates that the accuracy is statistically higher than a random guess. The accuracy values achieved by kNN in read histograms are significantly larger than a random guess proving that samples from the same category are well clustered. The accuracy for write histograms is not as high but still better than random guesses. In summary, our study shows that read and write histograms are useful features for detecting malicious websites.

2) *Dynamic Libraries*: We also build a histogram for dynamic library usage of websites. Similarly, the library histogram of a provenance graph is a key-value map where keys are names of dynamic libraries loaded by websites, and values mean how many times each library has been used. For instance, assuming a website uses A.dll twice and B.dll once, we build the library histogram for the site as {A.dll : 2, B.dll : 1}.

We visualize the library-histograms of benign and malicious websites with t-SNE in Fig. 8 for different combinations of browsers and OSes. We find that library histograms of malicious and benign websites are also well clustered. In Fig. 8(a), (c), and (d), we can see clusters that are either dominated by benign or

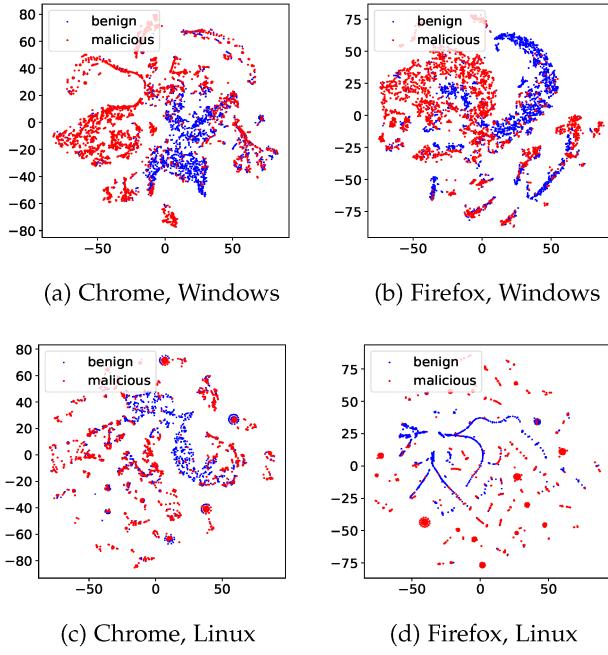


Fig. 7. Visualization of operation histograms.

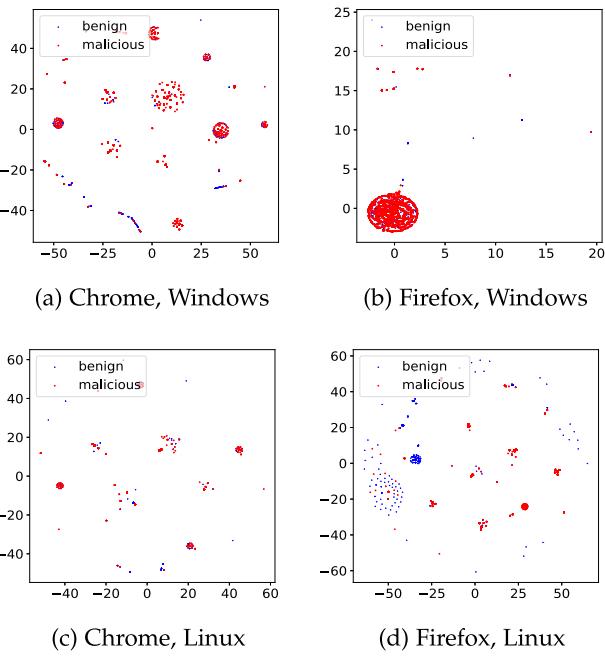


Fig. 8. Visualization of library histograms.

malicious websites. The only exception is Fig. 8(b), where we find the reason is that the Windows version of Firefox seems only to use a fixed set of dynamic libraries. Thus library histograms of all websites in the Windows version of Firefox are similar.

To statistically answer this research question, like Section IV-D1, we also use kNN to quantitatively evaluate how well library histograms can detect malicious websites. We use

the same protocol in Section IV-D1. The accuracy numbers and p-values for ten different experiments are shown in Table IV, which indicates that, except for the Windows version of Firefox, the prediction accuracy is from 64.8% to 68.3%. These numbers are statistically larger (with $p < 0.05$) than a random guess. In summary, except for the Windows version of Firefox, we find library histograms can be used as effective features for malicious website detection.

3) Operation Types: We then investigate whether malicious websites operate differently from benign ones by measuring the operation histogram from the provenance graph of each website. We consider the operation types listed in Table II. Similar to read and write histograms, we calculate the frequency of each operation type as a key-value map, where keys are operation types and values are frequencies of each operation type. For example, if a website reads files twice, forks processes once, and writes to files once, then the histogram of operation types is $\{read : 2, fork : 1, write : 1\}$.

The visualization of operation histograms with four configurations is shown in Fig. 7, where red represents malicious websites, and blue represents benign websites. We find that the operation histograms are also well clustered. For example, in Fig. 7(a), benign samples are mainly located in the center-left of the graph, while malicious samples are in the peripheral parts. Similarly, in Fig. 7(c), benign samples are mainly clustered in the central part while malicious websites are scattered in other parts. We can also observe clusters of either benign or malicious websites in Fig. 7(b) and (d).

To statistically answer this research question, we measure the prediction accuracy with kNN based on operation histograms with the same protocol in Section IV-D1 and repeat the experiment ten times. The results and p-values are reported in Table IV. The prediction accuracy of kNN on operation histograms ranges from 81.9% to 91.2% with p-values smaller than 0.05 in our study, which indicates that operation histograms can be used as effective features for malicious website detection.

4) Network: This section tries to answer whether malicious websites use IP addresses differently from benign websites. Since IP addresses can vary significantly, we cannot build the same histogram as we did in Section IV-D1. To address this challenge, we build a concentration vector C instead. Our intuition is that malicious websites are less organized. They may use more resources from third parties. Thus, the IP addresses of malicious websites may be more diffused than benign websites.

The concentration vector C is defined as $C = (S, I)$, where S is the number of subnets accessed by a website. For an IP address, We take the first two octets as the mask for the subnet. I is the number of IPs accessed by a website. The concentration vectors of all websites are visualized in Fig. 9, where the x -axis is S , the y -axis is I , red represents malicious websites, and blue represents benign websites. The blue line and the red line are the regression lines for benign and malicious websites, respectively. The green line is the baseline, which represents every IP from a unique subnet.

Fig. 9 shows that IPs of benign websites tend to be more organized. The ratio I/S is larger for benign websites than for

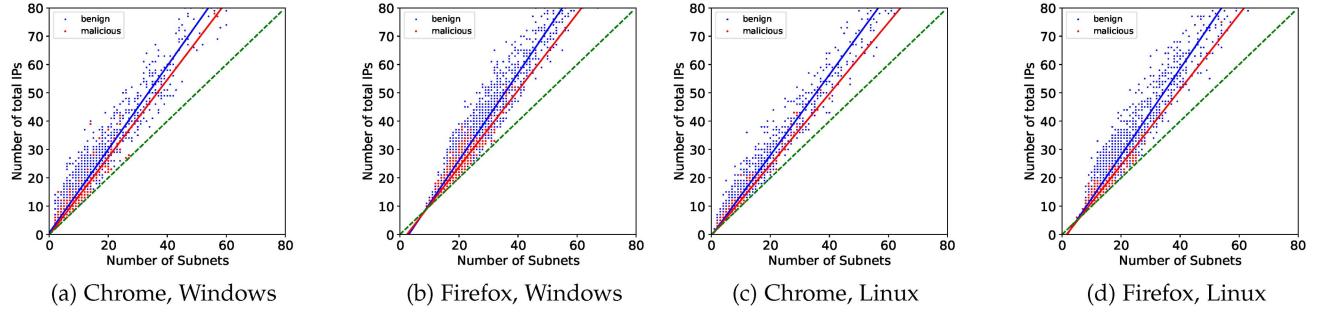


Fig. 9. The scatter graph of IP concentration vectors.

TABLE V
THE ACCURACY OF LOGISTIC REGRESSION CLASSIFIER ON IP CONCENTRATION VECTOR

	Chrome Windows	Firefox Windows	Chrome Linux	Firefox Linux
IP	72.8%	78.2%	80.8%	77.2%

P-values are all smaller than 0.05.

malicious websites. This result indicates that malicious websites tend to access IPs from random locations. A similar observation is also mentioned by Oest et al. [49].

To statistically evaluate the effectiveness of the network features, we also train a simple logistic regression classification model based on the concentration vector and perform the test on our data set as we did in Section IV-D1 and repeat the experiment ten times with different randomly sampled testing datasets. The result is shown in Table V, which shows that the concentration vector of IP addresses can classify malicious websites better than a random guess with p-values smaller than 0.05.

5) *Process*: We measure and compare the distribution of processes forked by browsers between malicious websites and benign websites. Our result shows there are no significant differences. The reason is that in our experiment, browsers only fork processes that belong to browser packages. We do not find any other processes. Yet, we notice that, in Chrome, malicious websites fork more software_reporter processes, which report crashes and errors, than benign websites. We hypothesize the reason is that malicious websites are not well-developed and have more bugs. However, the number of software_reporter processes is generally small, so it cannot make the difference statistically significant. As a result, we will not use this feature in our detection.

E. Conclusion on Semantic Features

Our empirical results show that malicious websites and benign websites are distinguishable in file read and write, network usage, operation types, and dynamic library usage. Thus we argue that file read histograms, write histograms, operation histograms, library histograms, and concentration vectors of IP addresses can be used as effective features for malicious website detection.

V. DETECTION TOOL

We design and implement a malicious website detection tool ProvWeb, which leverages our discoveries in Section IV to analyze system provenance graphs of browsers and detect whether browsers have visited malicious websites.

A. Architecture and Workflow

Fig. 10 presents the workflow. ProvWeb is an online provenance-based malicious website detection tool. Its input is provenance event streams collected from each browser instance. We leverage a similar technique in Wang et al. [19] for provenance collection. The output of ProvWeb is a binary classification on each of the browser instances, which indicates whether the instance is visiting a malicious website.

ProvWeb includes two main components. The first is feature extraction, which extracts the feature vectors from the provenance event stream without explicitly building and traversing provenance graphs. This algorithm allows ProvWeb to process provenance events in real time. The feature vectors include the features we discussed in Section IV. The second component is feature aggregation, which combines different features in the feature vector to detect malicious websites. The feature aggregation part is a machine learning model that is offline trained. However, ProvWeb uses the model to detect malicious websites in real time.

B. Feature Vector

Formally, we define our feature vector as $V = \langle N, FR, FW, L, OP \rangle$. In the feature vector, $N = \langle n_n, n_e, w, C \rangle$ is the numerical vector, where n_n is the number of nodes of a provenance graph, n_e is the number of edges, w is the width, and $C = (S, I)$ is the concentration vector of IP addresses. For other parts, FR is the file read histogram, FW is the file write histogram, L is the library histogram, and OP is the operation histogram. Details of elements in FR , FW , L , and OP and reasons for selecting these features are discussed in Section IV.

C. Feature Extraction

We first use the feature extraction step to build the feature vector from the provenance event stream. Calculating the width

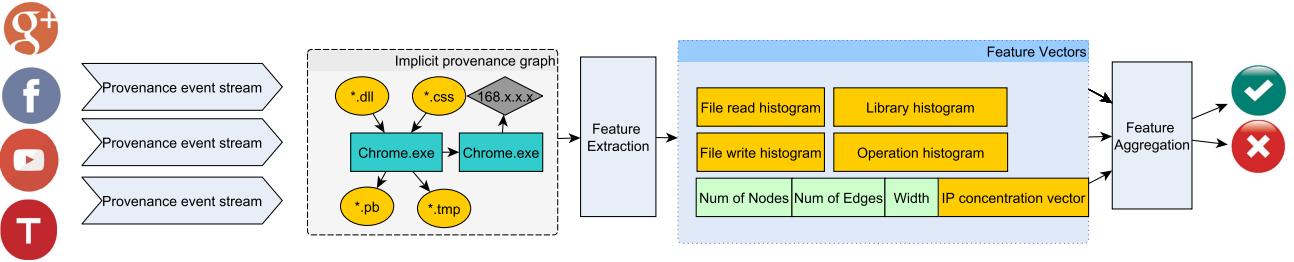


Fig. 10. The architecture of ProvWeb.

Algorithm 1: Feature Vector Building.

```

Data: Event stream  $E$  of a browser process
Result: Feature vector  $V$  of the process
1 initialize  $n_n$ ,  $n_e$ ,  $w$ , and  $C$  as zeros;
2 initialize histograms  $FR$ ,  $FW$ ,  $L$ , and  $OP$  as zeros;
3 initialize event depth map  $NDM$  as empty;
4 initialize  $W$  as zeros;
5 while  $E$  is not Empty do
6    $e = E.poll();$ 
7   if  $e.dst \notin NDM$  then
8      $NDM[e.dst].depth = NDM[e.src].depth + 1;$ 
9      $W[NDM[e.dst].depth] += 1;$ 
10  end
11   $n_e += 1;$ 
12  update  $OP$  based on  $e.optype$ ;
13  if  $e.dst$  is a file then
14    update  $FR$  or  $FW$  based on  $e.optype$ ;
15  end
16  if  $e.dst$  is a dynamic library then
17    update  $L$ ;
18  end
19  if  $e.dst$  is a new IP address then
20     $C.I += 1;$ 
21  end
22  if  $e.dst$  is a new subnet then
23     $C.S += 1;$ 
24  end
25 end
26  $n_n = \text{len}(NDM);$ 
27  $w = \max(W);$ 

```

of a provenance graph requires the structure information of the graph. The naive approach is to traverse the graph, which cannot be achieved online. In our approach, we propose an algorithm that directly calculates the feature vector without traversing the provenance graph explicitly.

Algorithm 1 shows our process of building the feature vector. Instead of maintaining the provenance graph, ProvWeb maintains two intermediate data structures, the node depth map NDM (line 3) and the width cache W (line 4). Formally, the node depth map is a key-value map $NDM = < N, depth >$, where N is a node in the provenance graph, and $depth$ is the depth of the node on the temporal spanning tree of the system provenance graph. The width cache is a key-value map

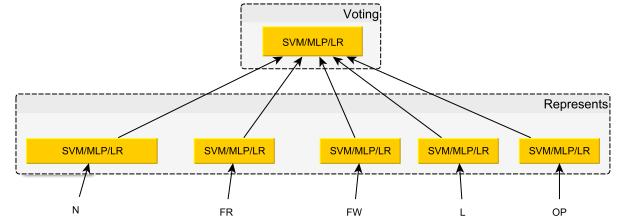


Fig. 11. The structure of feature aggregation.

$W = < depth, num >$. $W[k] = m$ means there are m nodes of which depths are k .

Our process to build the feature vector is then straightforward. ProvWeb takes each event as an edge in the provenance graph. For each event in the provenance stream, ProvWeb first adds the destination of the event to NDM and updates its depth if it is a new node (lines 7- 9). ProvWeb also updates the operation histogram OP (line 12). If the event is a file read or write event, ProvWeb updates FR and FW based on the operation type (lines 13 - 15). If the event represents a dynamic library loading, ProvWeb updates L . Then, if the event uses IP addresses, ProvWeb also updates the concentration vector of IP addresses. Finally, ProvWeb calculates n_n and w , which equals the size of the node depth map and the max value of the width cache, respectively (lines 20-22).

D. Feature Aggregation

During the feature aggregation step, the tool accepts the feature vectors and generates a probability of whether the given feature vectors represent a browser instance that visits malicious websites. At the high level, our model has two layers, as shown in Fig. 11. The first layer is the representation layer, which contains binary classification models for each of the features in the feature vector, respectively. The second layer is the voting layer, which combines the outputs of represents to generate the final output. This design is inspired by stacking in ensemble learning [50], which has been shown that can greatly improve the generalization of a single model [51].

ProvWeb draws two strengths from stacking and uses input attribute perturbation to improve stacking further. To be more specific, ProvWeb uses different sub-models in the representation layer, like stacking. As Krogh and Vedelsby [52] proposed, the higher the sub-learner's diversity is, the better

the ensemble model performs. Second, unlike the majority voting method used in typical ensemble learning algorithms, the combination strategy of ProvWeb is to use the "learning method" in the voting layer [51], [53] as stacking does. This learning-based voting layer attempts to minimize the weakness and maximize the strengths of every sub-model [54]. ProvWeb further improves stacking by feeding different sub-features into sub-models which provides different perspectives to observe the data, allowing a more robust model [55]. As we will evaluate later, ProvWeb has higher accuracy than other alternative techniques, such as simple majority voting and combining all features as input.

There can be multiple choices for the actual models used in both layers. In this paper, we implement several candidates and select them based on empirical measurement. For the representation layer, candidate classifiers are:

- The logistic regression model for each of the numerical features: n_n , n_e , w , and C , and the SVM model for histograms: FR , FW , OP and L .
- MLP model for the concatenated numerical vector $N = \langle n_n, n_e, w, C \rangle$ and SVM model for four histograms: FR , FW , OP and L .
- Logistic regression model for all both numerical features: n_n , n_e , w , and C and four histograms: FR , FW , OP and L .

For the voting layer, candidates are:

- The SVM model which accepts the output of all classifiers in the first layer
- The MLP model which accepts the output of all classifiers in the first layer
- The logistic regression model which accepts the output of all classifiers in the first layer

Formally, for the logistic regression model, we define each sample as (x_i, y_i) where x_i is one of the feature vectors and $y_i \in \{0, 1\}$ is the label of the given sample. $y_i = 1$ means the given sample is malicious and $y_i = 0$ means otherwise. Then the regression model is to minimize the cost function $\frac{1}{m} \sum (-y_i \log(h(Ax_i + b)) - (1 - y_i) \log(1 - h(Ax_i + b)))$, where m is the number of samples, $h()$ is the sigmoid function, and A and b are parameters for training.

To generate probabilities as the logistic regression model does, we apply Platt scaling [56] to linear SVM classifiers. Specifically, Platt scaling adds logistic regression to the output of the SVM model. Formally, the linear SVM model with Platt scaling has two parts. The first part is a normal linear SVM classifier. We define the input of the model as (x_i, y_i) where x_i is the input feature and $y_i \in \{-1, 1\}$ is the label of the given sample. $y_i = 1$ means the given sample is malicious and $y_i = -1$ means otherwise. The SVM is to maximize the target function $\max_{w,b} \min_{|w|} y_i(wx_i + b)$, where w and b are parameters for training. The second part is a logistic regression trained separately. In other words, our model minimizes the cost function: $\sum (-t_i \log(h(Af_i + b)) - (1 - t_i) \log(1 - h(Af_i + b)))$, where $t_i = \frac{y_i + 1}{2}$, f_i is the output of the linear SVM model, and A and b are trainable parameters.

The MLP models in both layers are the same. The basic model has three layers, one input layer, one hidden layer,

and one output layer. Formally, the MLP model is defined as follows: $x_i = \text{relu}(W_i \cdot X + b_i)$ $x_h = \text{relu}(W_h \cdot x_i + b_h)$ $x_o = \text{softmax}(W_o \cdot x_h + b_o)$, where X is the input feature vector, W_i , W_h , W_o , b_i , b_h , and b_o are trainable parameters, x_o is the final output.

VI. EVALUATION

To evaluate the performance of ProvWeb, we answer the following research questions:

- How accurate is ProvWeb?
- What is the runtime system overhead of ProvWeb?

A. Experiment Protocol

To evaluate ProvWeb, we further downloaded 800 new malicious website samples from VirusShare and PhishTank. We also collected 400 new benign websites from Alexa's top 10,000. In total, we have 1,200 new websites in our testing set. We use the data set mentioned in Section IV as the training set. Note that there is no overlap between our testing and training sets. We use a Ubuntu 18.04 desktop with an i7 CPU, and 32 GB memory to train the detection model and test run-time detection overhead of ProvWeb on the same machine.

B. Accuracy

To evaluate its accuracy, we compare ProvWeb with three baseline models: SVM, MLP, and Logistic Regression (LR). For baseline models, the input is the concatenation of all features. We measure the recall, precision, and F1 score of ProvWeb and the baseline models. The result is shown in Table VIII.

In our experiment, we find that all detection models have achieved decent accuracy in malicious website detection. The F1 scores of all models range from 82% to 99%. This number indicates that the features we collected are effective in detecting malicious websites. Particularly, ProvWeb performs consistently better than baseline models. The F1-scores of ProvWeb range from 93% to 99%, which are up to 11% higher than baseline models. This result also validates our algorithm design of ProvWeb.

1) Classifier Selection: In Section V-D, we designed three candidates for the classifiers in the representation layer and the voting layer, respectively. In this section, we also evaluate the F1-score of ProvWeb with different combinations of classifiers. The result is shown in Table VI, in which "X + Y" means we use classifier X for the numerical vector and Y for other feature vectors. Table VI shows that ProvWeb achieves the best accuracy when we use LR for numerical vectors, SVM for other feature vectors, and MLP for voting. We then use this configuration as the default configuration of ProvWeb.

Majority Voting: We also apply the simple majority voting in the final layer based on the classification results of the representation layer. Since we get class probabilities in the first layer, we choose the Soft Voting Classifier as the majority voting [57]. Soft voting classifier predicts the class label based on the argmax of the sums of the predicted probabilities. The result is shown in the rows of LR+SVM+Vote, LR+SVM+Vote,

TABLE VI
PERFORMANCE OF THE HIERARCHICAL MODEL WITH DIFFERENT CLASSIFIERS

F1 Voting Represents	Chrome, Windows			Firefox, Windows			Chrome, Linux			Firefox, Linux		
	SVM	MLP	LR	SVM	MLP	LR	SVM	MLP	LR	SVM	MLP	LR
LR + SVM	94.4%	93.7%	93.7%	94.9%	95.7%	95.1%	96.6%	96.8%	96.7%	99.7%	99.7%	99.7%
MLP + SVM	91.2%	92.0%	91.9%	95.4%	95.3%	95.2%	95.8%	95.8%	95.9%	98.6%	98.5%	98.5%
LR + LR	92.8%	93.5%	93.6%	93.3%	94.2%	92.3%	95.9%	95.7%	95.5%	99.6%	99.7%	99.7%

TABLE VII
PERFORMANCE OF THE HIERARCHICAL MODEL WITH DIFFERENT CLASSIFIERS WHEN THE RATIO OF TEST DATA AND TRAINING DATA IS 8:2

F1 Voting Represents	Chrome, Windows			Firefox, Windows			Chrome, Linux			Firefox, Linux		
	SVM	MLP	LR	SVM	MLP	LR	SVM	MLP	LR	SVM	MLP	LR
LR + SVM	92.4%	92.4%	92.2%	92.9%	93.5%	92.4%	95.5%	95.6%	95.5%	98.9%	99.2%	98.9%
MLP + SVM	89.2%	89.4%	89.3%	93.9%	93.9%	94.70%	92.1%	92.5%	93.9%	97.2%	97.8%	97.7%
LR + LR	91.9%	92.1%	92.1%	91.9%	92.2%	91.1%	94.8%	94.9%	94.8%	98.6%	99.3%	98.9%

TABLE VIII
ACCURACY OF DETECTION MODELS

	Chrome, Windows			Firefox, Windows			Chrome, Linux			Firefox, Linux		
	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
ProvWeb	95.1%	94.4%	93.7%	97.2%	94.3%	95.7%	97.6%	96.0%	96.8%	99.6%	99.7%	99.7%
SVM	84.4%	82.6%	83.5%	96.0%	94.6%	95.3%	92.1%	89.9%	91.0%	95.4%	91.1%	93.2%
MLP	84.3%	82.7%	83.5%	95.2%	94.6%	94.9%	94.2%	91.2%	92.7%	94.2%	93.1%	93.7%
LR	90.8%	75.7%	82.6%	96.2%	92.5%	94.3%	97.1%	85.7%	91.0%	92.4%	91.6%	92.0%

TABLE IX
ACCURACY OF DETECTION MODELS WHEN THE RATIO OF TEST DATA AND TRAINING DATA IS 8:2

	Chrome, Windows			Firefox, Windows			Chrome, Linux			Firefox, Linux		
	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
ProvWeb	91.3%	93.5%	92.4%	95.1%	92.0%	93.5%	96.2%	95.2%	95.6%	99.2%	99.1%	99.2%
SVM	83.5%	75.3%	79.2%	95.5%	91.2%	93.2%	95.7%	84.7%	89.9%	94.4%	90.9%	92.2%
MLP	84.1%	75.1%	79.3%	95.1%	92.2%	93.3%	91.3%	89.4%	90.3%	93.3%	92.1%	92.6%
LR	87.0%	74.6%	80.3%	96.0%	90.6%	93.3%	95.7%	83.8%	89.3%	91.3%	90.7%	91.0%

TABLE X
ACCURACY OF DETECTION MODELS INCLUDING PROVWEB AND MAJORITY VOTING STRATEGY

	Chrome, Windows			Firefox, Windows			Chrome, Linux			Firefox, Linux		
	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
ProvWeb	95.1%	94.4%	93.7%	97.2%	94.3%	95.7%	97.6%	96.0%	96.8%	99.6%	99.7%	99.7%
LR+SVM+Vote	98.5%	85.8%	91.7%	98.5%	81.8%	89.5%	99.4%	75.7%	86.1%	99.2%	99.8%	99.4%
MLP+SVM+Vote	97.5%	86.9%	91.9%	98.4%	88.6%	93.5%	98.2%	82.8%	90.6%	99.4%	99.7%	99.5%
LR+LR+Vote	99.7%	79.5%	88.5%	99.8%	74.1%	85.1%	97.9%	76.7%	86.8%	99.6%	99.2%	99.4%

and LR+SVM+Vote in Table X. Particularly, LR+SVM+Vote means LR for the numerical vector and SVM for other feature vectors, and Soft Voting Classifier for the voting layer. The same goes for LR+SVM+Vote and LR+SVM+Vote. We find that simple majority voting is relatively less effective than our learning-based approach.

C. Generalization Analysis

Considering that malicious websites can evolve over time and our samples can not enumerate all types of malicious websites, in this section, we try to evaluate the effectiveness of ProvWeb in detecting new malicious websites. To this end, we follow common protocols in the machine learning community to

evaluate the generalizability of features and supervised learning models [58]. Our evaluation protocol for generalizability is as follows.

First, we visualize the combined feature vectors, which is the concatenation of all input vectors, of the websites in the dataset we used in Section IV, with t-SNE [45]. This experiment intuitively shows how the combined feature vector can be used to detect malicious websites.

Second, we run the cluster analysis [59], [60] to evaluate how well-unsupervised methods can detect malicious websites. We adopt three popular unsupervised clustering algorithms [61]: K-means [62], Gaussian Mixture [63], and Spectral Clustering [64]. These three clustering algorithms are totally different in principle. To be more specific, we set the number of clusters

TABLE XI
CROSS-VALIDATION: EVALUATING MODEL GENERALIZATION PERFORMANCE

	Chrome, Windows		Firefox, Windows		Chrome, Linux		Firefox, Linux	
	Mean	Standard Deviation	Mean	Standard Deviation	Mean	Standard Deviation	Mean	Standard Deviation
ProvWeb	93.3%	0.008	94.5%	0.010	94.8%	0.005	99.6%	0.002
SVM	81.6%	0.012	94.9%	0.010	89.3%	0.01	93.8%	0.01
MLP	81.3%	0.014	94.8%	0.007	90.7%	0.009	93.9%	0.01
LR	77.7%	0.017	93.9%	0.009	86.6%	0.008	93.8%	0.01

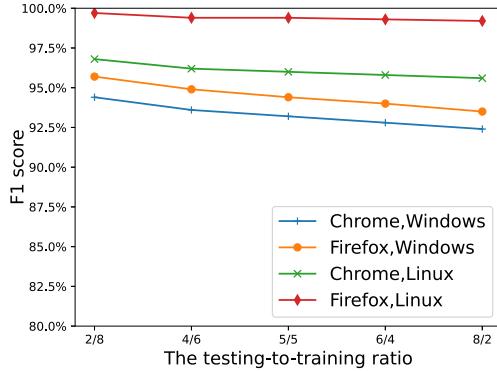


Fig. 12. Performance curve when the testing-to-training ratio changes from 2:8 to 8:2.

to two. We use them to evaluate how the feature vector detects malicious websites from different angles and avoid possible bias of using one single method.

In this experiment, we first visualize the clustering result with t-SNE to intuitively evaluate how well the clustering algorithms cluster data. Then, we report the average purity [65] of each clustering algorithm, which numerically represents how well can a clustering algorithm cluster samples from the same type into the same cluster. We also tested whether the purity value is higher than 50%, which is the number of random guesses, with the Kolmogorov-Smirnov [46] test and reported the p-values.

Third, we use a nested 10 cross-validation [58] to evaluate the generalizability of ProvWeb. A nested 10 cross-validation is a classic method to evaluate the generalizability of machine learning models [58]. To this end, we randomly split the training data into ten folds. To be more specific, the nested cross-validation is to nest the hyperparameter optimization procedure under the model selection procedure. It overcomes the optimistically biased evaluation introduced by using the same cross-validation procedure and dataset for tuning and selecting a model.

Fourth, to show that ProvWeb is generalizable when the training set is much smaller than the testing set, we adjust the size of the training set. Specifically, we adjust the testing-to-training ratio from 2:8 to 8:2 gradually. Then, we evaluate whether the model performance has significantly decreased when the training set gets smaller and smaller.

1) *Results:* For the first experiment, we show the visualization results of the four different configurations in Figs. 13(a), (c), 14(a), and (c), respectively. We find benign samples are mainly well clustered while malicious websites are scattered in other parts. This result shows that the combined feature vector is sufficient to separate malicious websites from benign websites.

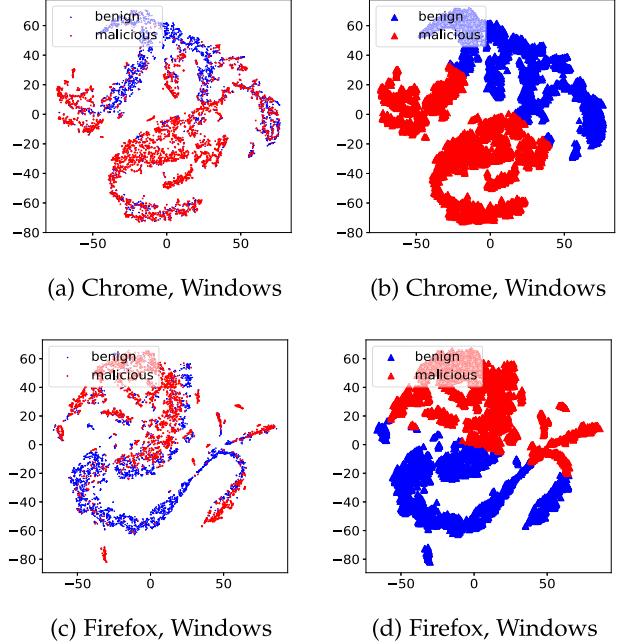


Fig. 13. Visualization of Windows.

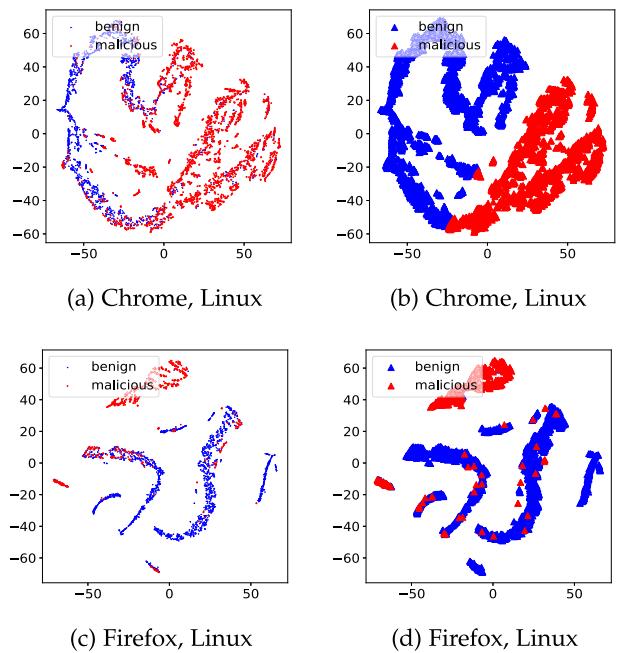


Fig. 14. Visualization of Linux.

TABLE XII
PURITY OF DIFFERENT CLUSTERING CLASSIFIERS ON THE CONCATENATION FEATURE

	Chrome Windows	Firefox Windows	Chrome Linux	Firefox Linux
Kmeans	66.5%	69.6%	67.8%	71.4%
Spectral Clustering	58.5%	60.1%	59.8%	71.6%
Gaussian Mixture	70.6%	67.3%	69.9%	87.7%

P-values are all smaller than 0.05.

For the second experiment, we plot the visualizations in Figs. 13(b), (d), 14(b), and (d). These graphs intuitively show that clustering algorithms can detect malicious websites based on the combined feature vector. We report the average purity number of different clustering models and configurations, along with the p-values of the Kolmogorov-Smirnov test, in Table XII. In general, Kmeans achieves the best Purity for the Windows version of Firefox, while Gaussian Mixture is the best for other browsers. The purity is from 58.5% to 87.7%. The p-values are all smaller than 0.05, which means the purity values are statistically higher than random guesses.

For the third experiment, we show the average validation accuracy and its standard deviation in Table XI. We can see the average score is close to the results shown in Table IX. The standard variance of 10 evaluations is less than 0.02, which means our supervised learning-based methods are robust to the different splits of data.

For the last experiment, we report the results in Tables VII, IX and Fig. 12. Though the size of the training set is only a quarter of the size of the test set, the overall metrics of the new model are only two percentage points lower than the previous model. It means even a relatively small dataset can train a good model based on our features. It should be noted that when the testing-to-training ratio is 8:2, there are still more than 1,000 training samples.

Discussion: Our experiments show that it is possible to detect malicious websites with unsupervised algorithms. However, it is well-known that unsupervised algorithms are ineffective in learning high dimensional data similar to our combined feature vector [60]. Thus, the unsupervised models in our experiments are less accurate than ProvWeb.

D. System Overhead

To understand the overhead of ProvWeb, we measure the training time for offline model training and the CPU and memory usage during online detection.

1) *Training Time:* The model of ProvWeb is trained offline. The training time for ProvWeb includes two parts: (1) time for abstracting features from system provenance data; (2) time for training the detection model. In our study, the time cost for abstracting features from 8,000 websites is 20 hours. The main part (99%) for this time overhead is from iterating over the system provenance database on the disk. We believe that by carefully designing hashing policies and data schema, this time could be substantially reduced. We leave this optimization as our future work. The average time for training the detection model in our study is 16 minutes on 8,000 websites. Note that

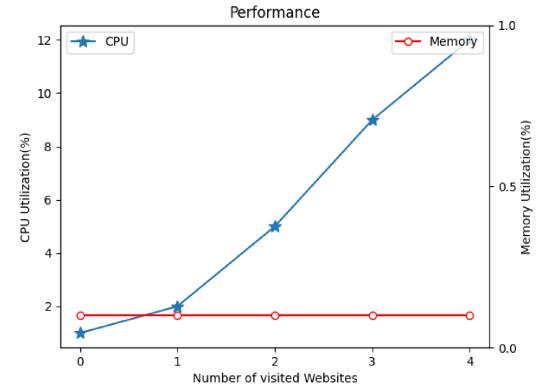


Fig. 15. Detection overhead per website.

the overhead for training is a one-time cost and the model does not require retraining unless we need to upgrade it.

2) *Real-Time Detection Overhead:* We first measure CPU and memory usage of our tool ProvWeb. In Windows, the overhead of both CPU and memory are kept below 1%, which seems to be less impacted by the number of visited websites. Fig. 15 presents the Linux result, which shows how CPU and memory usage goes w.r.t. the number of visited websites. We found that CPU usage increases when more websites are visited concurrently. More specifically, a simple website will cause a 3% CPU usage increase, while a complicated one will cause a 5%~7% CPU usage overhead. In our deeper investigation, we find that more than 99% of CPU usage overhead is caused by sysdig itself. Similar overheads are also reported in previous provenance data analysis work [17], [20], [21]. The memory usage maintains within 0.1% regardless of the number of websites visited. We then measure the detection time. On average, ProvWeb takes only 0.32 ms to make a prediction.

In summary, the system overhead of ProvWeb is mainly caused by the underline system provenance collection tool, which contributes more than 99% of the runtime overhead. We prevision that the system overhead of ProvWeb can be reduced with an optimized provenance collection tool. However, we leave the development of such tools to future work. Nevertheless, the detection overhead of ProvWeb is still comparable to other system provenance techniques [17], [20], [21].

VII. RELATED WORK

To the best of our knowledge, this paper is the first systematic study on malicious websites from the perspective of system provenance analysis. Conventionally, people rely on techniques such as URL blacklisting [6], [66], static code analysis [8], and dynamic code analysis [10] to identify threats from malicious websites. Despite the effectiveness of conventional techniques, our study and ProvWeb are orthogonal to them.

A. URL Blacklist

Static blacklisting is ineffective in detecting new malicious websites because it suffers from the difficulty of building an exhaustive list of all malicious URLs [67]. To alleviate the

limitation, dynamic approaches based on URL features are proposed [4], [6], [66], [68], [69], [70].

Recently, researchers also apply deep learning techniques for malicious URL detection. Shibahara et al. applied a convolutional neural network to detect malicious URL sequences [71]. Zhang et al. proposed a semi-supervised model to detect malicious URLs [72]. Sahoo et al. conducted a survey on machine learning-based malicious URL detection techniques [73]. Despite these progresses, URL blacklisting is still incomplete and error-prone. Adversaries can use a new URL that has never been detected or directly uses benign domains. As reported, 40% of malicious web pages are in good domains [7]. Adversaries may hijack a trusted website with tools like Darkleech [74]. As reported, in 2012–2013, 40,000 websites were infected by Darkleech [75]. The well-known URL blacklist PhishTank also reports thousands of new malicious URLs per day [38]. The limitations of URL blacklisting indicate that other orthogonal techniques are needed to secure web browsing.

B. Code Analysis

People also rely on code analysis techniques to detect malicious websites. Prophiler [8] and Cujo [76] use static HTML features and JavaScript features to predict if a web page is malicious or benign. JStap [77] uses static features of Abstract Syntax Tree, Control Flow Graph, and Dependency Graph to detect malicious websites. The main limitation of above mentioned static-based approaches is that they are not robust against obfuscation and camouflaging [9].

Revolver [11] dynamically monitors the AST of JavaScript in the JS engine of browsers and checks whether the dynamically generated AST of JavaScript is benign. JSGraph [10], modifies the browsers to record the DOM manipulation actions of web pages so that it can detect web-based attacks. VisibleV8 [12] modifies the Chrome engine to log native functions of JavaScript code. Nazca [78] aims to detect malware download requests in the network. It builds models based on web traffic. There are also many other earlier approaches [79], [80], [81], [82], [83], but they are all orthogonal to our work.

C. System Provenance Analysis

System provenance analysis has been applied in many areas recently. Han et al. proposed Frappuccino, which uses system provenance data to detect faults for Platform as a Service (PaaS) users [84]. Xie et al. proposed an intrusion detection system based on system provenance data mining [85]. SIGL applies tree-based LSTM on system provenance data to detect malicious package installation [20]. Wang et al. proposed a technique to detect file-less malware with system provenance data [19].

A substantial number of techniques are proposed to address Advanced Persistent Threats (APT) attacks. King et al. proposed the idea of building dependency graphs on system provenance data [13]. SPADE is one of the first practical tools that build dependency graphs on system provenance data [86]. Based on the dependency graph, Hossain et al. proposed SLEUTH to track APT attacks in real time [41]. The same authors later proposed a tag alternative propagation algorithm to address the

path explosion problem of system provenance analysis [87]. Similarly, Gao et al. proposed SAQL, a stream based language to support real time system provenance data queries [14]. Gui et al. proposed a new system provenance querying system [15]. Milajerdi et al. proposed HOLMES that improves SLEUTH by combining detection with APT attack tracking [88]. Besides applying system provenance analysis to detect attacks, researchers also proposed techniques to improve the system provenance data collection systems [89], [90], [91] and storage systems [92], [93], [94].

Although system provenance analysis has shown its usefulness in multiple security problems, none of the existing approaches applies system provenance analysis to malicious website detection. To the best of our knowledge, this work is the first systematic research on malicious websites from the perspective of system provenance analysis.

VIII. DISCUSSION

In this section, we discuss possible threats to the validity of our study.

A. Datasets

To ensure our study can represent common types of malicious websites, we perform experiments on two independent data sources: VirusShare [37] and PhishTank [38]. The former is a collection of malicious HTML and JavaScript files, and the latter is a collection of phishing websites. Our data are collected without any selection or filtering. We collect benign samples from Alexa's top 10,000 websites, which represent the most accessed websites worldwide. We believe this data set could reflect the distribution of recently discovered malicious websites and the most popular benign websites. Adversaries may obfuscate or camouflage the code of malicious websites. However, we do not expect code obfuscation and camouflaging could substantially change our conclusion since our study is based on dynamic analysis.

We measure the provenance graph of each website on two mainstream browsers, Chrome and Firefox, which represent 81% of the world market, and two mainstream OSes, Windows 7 and Ubuntu 18.04. We believe that the browsers and OSes we used in this study could represent the current ecosystem.

B. Anti-VM

Our experiments are made in Virtualbox driven by Cuckoo Sandboxes. Advanced malware may apply anti-VM techniques to avoid exposing malicious behaviors in virtual machines, which may affect the validity of our study. While anti-VM techniques are popular in other types of malware, they are less common in malicious websites because browsers do not expose most of the system information required by popular anti-VM techniques [95]. We have also manually checked our data set and spotted no samples with anti-VM features. We believe the result of our study is not significantly altered by anti-VM techniques. Further, our study does not only look for malicious behaviors but also looks for statistical differences in general behaviors between

malicious and benign websites. To change statistical patterns of general behaviors, adversaries have to spend more efforts to substantially change the structure of their websites, which could further reduce the motivation to deploy evasive techniques. We leave the technique to address malicious websites using anti-VM techniques as future work.

C. Comparison to Other Techniques

ProvWeb is the very first provenance analysis technique for malicious website detection. We do not directly compare ProvWeb to existing techniques because they are fully orthogonal. System provenance analysis may help alleviate the limitations of conventional techniques. Unlike URL blacklist and static code analysis, system provenance analysis is robust against URL changing and code obfuscation. Compared to conventional dynamic analysis techniques, system provenance analysis does not require modifications to browsers so that it can be applied to broader scenarios. We believe ProvWeb can be applied together with existing techniques and help enhance web security in general.

D. Limitations

ProvWeb is based on the provenance analysis technique. The prerequisite is that the collected provenance data must be integral and trustworthy. However, attackers may engage in anti-forensic activities to cover their tracks, including the erasure and manipulation of provenance data [96], [97], [98]. For example, the asynchronous multi-producer single-consumer architecture of existing system provenance collectors imposes a lag between the generation and consumption of a system call event. Paccagnella et al. [97] proposed an attack that deletes system call events generated by malicious processes from the ring buffers before they have been submitted to the userspace component. What's worse, the provenance collector may voluntarily drop provenance events when the system workload is very high, which is reported by CVE-2019-833 [98]. In this paper, we assume that the storage and transmission of the provenance data are trusted. We also assume the integrity of the provenance data as well as the provenance collector.

IX. CONCLUDING REMARKS

In this paper, we conduct the very first systematic study on system provenance data of malicious websites. This study closes the gap between system provenance analysis and malicious web detection. Having a provenance-based technique enables non-intrusive malicious website detection. Our study shows that the system provenance data of malicious websites are substantially different from the system provenance data of benign websites. Based on our measurement study, we summarize eight system provenance-based features for malicious website detection and build a detection tool ProvWeb. Compared with conventional techniques, ProvWeb is non-intrusive as it does not require any modifications to browsers. In our evaluation, ProvWeb can achieve an F1 score of 93.7% ~ 99.7% for the four combinations of browsers and OSes (Windows Chrome, Windows Firefox, Linux Chrome, Linux Firefox). Our study demonstrates that

system provenance analysis offers a non-intrusive and more robust mechanism to detect malicious websites effectively. As it has the potential to improve detection accuracy further if combined with other existing techniques, we will explore more possibilities in system provenance-based malicious website detection techniques in the future.

REFERENCES

- [1] Kaspersky, "Web threats," 2019. [Online]. Available: <https://usa.kaspersky.com/resource-center/threats/web>
- [2] McAfee, "Why web-based malware is the most serious threat to your business," 2017.
- [3] JOSH LAKE, "What is a drive-by download and how can it infect your computer?," 2019. [Online]. Available: <https://www.comparitech.com/blog/information-security/drive-by-download/>
- [4] A. Oest et al., "PhishTime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 379–396.
- [5] M. Nadeau, "What is Cryptojacking? How to prevent, detect, and recover from it," 2020. [Online]. Available: <https://www.csomisc.com/article/3253572/whatiscryptojackinghow-to-prevent-detect-and-recover-from-it.html>
- [6] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proc. ACM Workshop Recurring Malcode*, Jan. 22, 2021, Art. no. 1.
- [7] S. Sjouwerman, "[Heads-up] 40 percent of malicious URLs found on good domains. YIKES," 2019. [Online]. Available: <https://blog.knowbe4.com/heads-up-40-percent-of-malicious-urls-found-on-good-domains.-yikes>
- [8] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proc. 20th Int. Conf. World Wide Web*, 2011, pp. 197–206.
- [9] A. Fass, M. Backes, and B. Stock, "HideNoSeek: Camouflaging malicious JavaScript in benign ASTs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1899–1913.
- [10] B. Li, P. Vadrevu, K. H. Lee, and R. Perdisci, "JSgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser JavaScript executions," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
- [11] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 637–652.
- [12] J. Jueckstock and A. Kapravelos, "VisibleV8: In-browser monitoring of JavaScript in the wild," in *Proc. Internet Meas. Conf.*, 2019, pp. 393–405.
- [13] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proc. 19th ACM Symp. Operating Syst. Princ.*, 2003, pp. 223–236.
- [14] P. Gao et al., "SAQL: A stream-based query system for real-time abnormal system behavior detection," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 639–656.
- [15] J. Gui et al., "APTrace: A responsive system for agile enterprise level causality analysis," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1701–1712.
- [16] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: Real-time APT detection through correlation of suspicious information flows," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 1137–1152.
- [17] W. U. Hassan et al., "NoDoze: Combatting threat alert fatigue with automated provenance triage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019.
- [18] S. Wang et al., "Attentional heterogeneous graph neural network: Application to program reidentification," 2019.
- [19] Q. Wang et al., "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proc. Symp. Netw. Distrib. Syst. Secur.*, 2020.
- [20] X. Han et al., "SIGL: Securing software installations through deep graph learning," 2020.
- [21] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: Runtime provenance-based detector for advanced persistent threats," 2020.
- [22] S. Kim, J. Kim, and B. Kang, "Malicious URL protection based on attackers' habitual behavioral analysis," *Comput. Secur.*, vol. 77, pp. 790–806, 2018.
- [23] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Tutorial and critical analysis of phishing websites methods," *Comput. Sci. Rev.*, vol. 17, pp. 1–24, Jan. 19, 2021.
- [24] PhishTank | join the fight against phishing, 2021. [Online]. Available: <http://phishtank.org/index.php>

- [25] Forrest Stroud, "Cryptomining malware," 2020. [Online]. Available: <https://www.webopedia.com/TERM/C/cryptomining-malware.html#:~:text=Cryptomining%20malware%2C%20or%20cryptocurrency%20mining,without%20a%20user%27s%20explicit%20permission>
- [26] C. Cimpanu, "Coinhive cryptojacking service to shut down in march 2019," 2019. [Online]. Available: <https://www.zdnet.com/article/coinhive-cryptojacking-service-to-shut-down-in-march-2019/>
- [27] The Monero Project, 2021. [Online]. Available: <https://www.getmonero.org/index.html>
- [28] Magecart Malware, Jun. 06, 2020. [Online]. Available: <https://sucuri.net/guides/website-malware/#:%E2%88%BC:text%20=What%20are%20the%20different%20types%20of%20website%20malware%3F.8.%20Credit%20card%20st%20ealers%20%26%20Ecommerce%20malware%20>
- [29] Magecart Malware, Jun. 16, 2022. [Online]. Available: <https://malpedia.caad.fkie.fraunhofer.de/details/js.magecart>
- [30] Supply-Chain Attacks, May 27, 2022. [Online]. Available: <https://www.darkreading.com/application-security/third-party-scripts-websites-broad-open-attack-vector>
- [31] SEO Spam, Jun. 10, 2020. [Online]. Available: <https://www.clickcease.com/blog/what-is-seo-spam/>
- [32] B. Eshete, A. Villaflorita, and K. Weldemariam, "Malicious website detection: Effectiveness and efficiency issues," in *Proc. 1st SysSec Workshop*, 2011, pp. 123–126.
- [33] H. Shahriar and M. Zulkernine, "Phishtester: Automatic testing of phishing attacks," in *Proc. 4th Int. Conf. Secure Softw. Integration Rel. Improvement*, 2010, pp. 198–207.
- [34] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "POIROT: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1795–1812.
- [35] Y. Liu et al., "Towards a timely causality analysis for enterprise security," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
- [36] S. Wang et al., "Heterogeneous graph matching networks for unknown malware detection," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 3762–3770.
- [37] VirusShare.com, 2021. [Online]. Available: <https://virusshare.com/>
- [38] PhishTank > developer information, 2020. [Online]. Available: https://www.phishtank.com/developer_info.php
- [39] Amazon, "Alexa top sites," 2020. [Online]. Available: https://aws.amazon.com/marketplace/pp/B07QK2XWNV?qid=1555346599089&sr=0-1&ref_=srh_res_product_title
- [40] D. Oktavianto and I. Muhandianto, *Cuckoo Malware Analysis*. Birmingham, U.K.: Packt Publishing, 2013.
- [41] M. N. Hossain et al., "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 487–504.
- [42] T. Wang, "High precision open-world website fingerprinting," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 152–167.
- [43] R. Conroy, "What hypotheses do "nonparametric" two-group tests actually test?," *Statist. J.*, vol. 12, pp. 182–190, 2012.
- [44] W. Conover, *Practical Nonparametric Statistics*, 3rd ed., ser. Wiley series in probability and statistics. New York, NY, USA: Wiley, 1999.
- [45] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 86, pp. 2579–2605, 2008.
- [46] R. Simard and P. L'Ecuyer, "Computing the two-sided Kolmogorov-Smirnov distribution," *J. Statist. Softw.*, vol. 39, no. 11, pp. 1–18, 2011.
- [47] How would you determine whether a classifier is significantly better than random guessing? Oct. 18, 2012. [Online]. Available: <https://www.researchgate.net/post/How-would-you-determine-whether-a-classifier-is-significantly-better-than-random-guessing>
- [48] W. U. Hassan et al., "This is why we can't cache nice things: Lightning-fast threat hunting using suspicion-based hierarchical storage," in *Proc. Annu. Comput. Secur. Appl. Conf.*, 2020, pp. 165–178.
- [49] A. Oest, Y. Safaei, A. Doupe, G.-J. Ahn, B. Wardman, and K. Tyers, "PhishFarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists," in *Proc. IEEE Symp. Secur. Privacy*, Oct. 07, 2020, pp. 1344–1361.
- [50] T. G. Dietterich et al., "Ensemble learning," *Handbook Brain Theory Neural Netw.*, vol. 2, pp. 110–125, 2002.
- [51] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.
- [52] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Proc. Adv. Neural Inf. Process. Syst.*, G. Tesauro, D. Touretzky, and T. Leen, Eds., MIT Press, 1994, pp. 231–238.
- [53] Stack Ensemble Learning, Mar. 25, 2022. [Online]. Available: <https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/>
- [54] How to use "model stacking" to improve machine learning predictions, Jul. 15, 2021. [Online]. Available: <https://medium.com/geekculture/how-to-model-stacking-to-improve-machine-learning-predictions-d113278612d4>
- [55] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.
- [56] J. Platt et al., "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Adv. Large Margin Classifiers*, vol. 10, no. 3, pp. 61–74, 1999.
- [57] Soft Voting Classifier, Sep. 07, 2020. [Online]. Available: <https://vitalflux.com/hard-vs-soft-voting-classifier-python-example/>
- [58] MML Book, "Mathematics for machine learning," in *Mathematics for Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2020, pp. 263–264.
- [59] M. L. D. S. Brian, S. Everitt, and S. Landau, "An introduction to classification and clustering," in *Cluster Analysis*, 5th Ed. Hoboken, NJ, USA: Wiley, 2011.
- [60] M. Steinbach, L. Ertöz, and V. Kumar, *The Challenges of Clustering High Dimensional Data*. Berlin, Germany: Springer, 2004, pp. 273–309.
- [61] Clustering Algorithms in Machine Learning, Sep. 21, 2020. [Online]. Available: <https://www.freecodecamp.org/news/8-clustering-algorithms-in-machine-learning-that-all-data-scientists-should-know/>
- [62] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means clustering algorithm," *Appl. Statist.*, vol. 28, no. 1, pp. 100–108, 1979. [Online]. Available: <http://dx.doi.org/10.2307/2346830>
- [63] D. N. Geary, "Mixture Models: Inference and Applications to Clustering," *J. Roy. Stat. Soc. Ser. A*, vol. 152, no. 1, pp. 126–127, Jan. 1989. [Online]. Available: <https://ideas.repec.org/a/bla/jorssa/v152y1989i1p126-127.html>
- [64] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proc. 14th Int. Conf. Neural Inf. Process. Syst. Natural Synthetic*, Cambridge, MA, USA: MIT Press, 2001, pp. 849–856.
- [65] Evaluation of Clustering:Purity, Apr. 07, 2009. [Online]. Available: <https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html>
- [66] C. Whittaker, B. Ryner, and M. Nazif, "Large-scale automatic classification of phishing pages," 2010.
- [67] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang, *An Empirical Analysis of Phishing Blacklists*. Pittsburgh, PA, USA: Carnegie Mellon Univ., 2009.
- [68] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious URLs," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, Sep. 21, 2020, Art. no. 1245.
- [69] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious URLs: An application of large-scale misc learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, Sep. 21, 2020, pp. 1–8.
- [70] A. Oest et al., "Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 361–377.
- [71] T. Shibahara et al., "Malicious URL sequence detection using event denoising convolutional neural network," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–7.
- [72] Y.-L. Zhang et al., "POSTER: A PU learning based system for potential malicious URL detection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 2599–2601.
- [73] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malicious URL detection using machine learning: A survey," 2020.
- [74] B. Duncan, "Campaign evolution: Darkleech to pseudo-Darkleech and beyond," 2016. [Online]. Available: <https://unit42.paloaltonetworks.com/unit42-campaign-evolution-darkleech-to-pseudo-darkleech-and-beyond/>
- [75] DAN GOODIN, "Rampant Apache website attack hits visitors with highly malicious software," 2013. [Online]. Available: <https://arstechnica.com/information-technology/2013/07/darkleech-infects-40k-apache-site-addresses/>
- [76] K. Rieck, T. Krueger, and A. Dewald, "Cujo: Efficient detection and prevention of drive-by-download attacks," in *Proc. 26th Annu. Comput. Secur. Appl. Conf.*, 2010, pp. 31–39.
- [77] A. Fass, M. Backes, and B. Stock, "JStap: A static pre-filter for malicious JavaScript detection," in *Proc. ACM 35th Annu. Comput. Secur. Appl. Conf.*, 2019, pp. 257–269.

- [78] L. Invernizzi et al., "Nazca: Detecting malware distribution in large-scale networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
- [79] P. Ratanaworabhan, V. B. Livshits, and B. G. Zorn, "Nozzle: A defense against heap-spraying code injection attacks," in *Proc. USENIX Secur. Symp.*, 2009, pp. 169–186.
- [80] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "Zozzle: Low-overhead mostly static JavaScript malware detection," in *Proc. USENIX Secur. Symp.*, 2011, pp. 3–3.
- [81] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious JavaScript code," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 281–290.
- [82] B. Eshete, "Effective analysis, characterization, and detection of malicious web pages," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 355–360.
- [83] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song, "A symbolic execution framework for JavaScript," in *Proc. IEEE Symp. Secur. Privacy*, 2010, pp. 513–528.
- [84] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer, "FRAPuccino: Fault-detection through runtime analysis of provenance," in *Proc. 9th USENIX Workshop Hot Top. Cloud Comput.*, 2017, pp. 18–18.
- [85] Y. Xie, D. Feng, Z. Tan, and J. Zhou, "Unifying intrusion detection and forensic analysis via provenance awareness," *Future Gener. Comput. Syst.*, vol. 61, pp. 26–36, 2016.
- [86] A. Gehani and D. Tariq, "SPADE: Support for provenance auditing in distributed environments," in *Proc. ACM/IFIP/USENIX Int. Conf. Distrib. Syst. Platforms Open Distrib. Process.*, 2012, pp. 101–120.
- [87] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1139–1155.
- [88] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: Real-time APT detection through correlation of suspicious information flows," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 1137–1152.
- [89] Y. Ji et al., "Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1705–1722.
- [90] T. Pasquier et al., "Runtime analysis of whole-system provenance," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1601–1616.
- [91] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy whole-system provenance for the Linux kernel," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 319–334.
- [92] K. H. Lee, X. Zhang, and D. Xu, "LogGC: Garbage collecting audit log," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 1005–1016.
- [93] M. N. Hossain et al., "Dependence-preserving data compaction for scalable forensic analysis," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1723–1740.
- [94] Y. Tang et al., "NodeMerge: Template based efficient data reduction for Big-Data causality analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1324–1337.
- [95] P. Chen, C. Huygens, L. Desmet, and W. Joosen, "Advanced or not? A comparative study of the use of anti-debugging and anti-VM techniques in generic and targeted malware," in *ICT Systems Security and Privacy Protection*, J.-H. Hoepman and S. Katzenbeisser, Eds., Berlin, Germany: Springer, 2016, pp. 323–336.
- [96] CAPEC-268: Audit log manipulation, Aug. 17, 2020. [Online]. Available: <https://capec.mitre.org/data/definitions/268.html>
- [97] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1551–1574.
- [98] M. Stemm, "CVE-2019-8339, a falco capacity related vulnerability," May 19, 2019 [Online]. Available: <https://sysdig.com/blog/cve-2019-8339-falco-vulnerability/?msclkid=4c2c25afa9b511ec815c58c5f800beec>



Peng Jiang received the BS degree in electrical engineering from Southeast University in July 2016 and the MS degree in electrical and computer engineering from Cornell University in 2018. Currently, he is working toward the PhD degree with the Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University. His research interests include security and system.



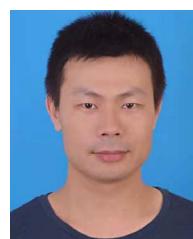
Jifan Xiao received the BS degree in computer science from Peking University in July 2021. Currently, he is working toward the PhD degree with the Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University. His research interests include system security and network security.



Ding Li (Member, IEEE) is an assistant professor with the Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University. His research interests include program analysis and system provenance analysis. Particularly, he is interested in applying machine learning techniques to make program analysis and system provenance analysis more efficient, and accurate.



Hongyi Yu received the degree of master's of Science in advanced computer science from the University of Essex in November 2019. Currently, he works with the Advanced Institute of Information Technology as a Big Data analysis engineer in the field of network security.



Yu Bai received the BE degree in communication and information systems from the PLA Electronic Engineering Institute in July 2012. Currently, he is the deputy director of the laboratory with the Advanced Institute of Information Technology (AIIT), Peking University. His research interests include information security and machine learning.



Yao Guo (Member, IEEE) received the PhD degree in computer engineering from the University of Massachusetts at Amherst in 2007. He is currently a full professor with the Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University. His general research interests include operating systems, mobile computing, and applications, low-power design, and software engineering.



Xiangqun Chen received the BS and MS degrees in computer science from Peking University. She is currently a professor with the Key Laboratory of High-Confidence Software Technologies (MOE), School of Computer Science, Peking University. Her research interests include operating systems and software engineering.