Efficient Multi-Scale Domain Detection

Emily Patel, Ryan Thompson
Massachusetts Institute of Technology, Cambridge, USA

Abstract—As world wild internet has non-stop developments, making profit by lending registered domain names emerges as a new business in recent years. Unfortunately, the larger the market scale of domain lending service becomes, the riskier that there exist malicious behaviors or malwares hiding behind parked domains will be. Also, previous work for differentiating parked domain suffers two main defects: 1) too much data-collecting effort and CPU latency needed for features engineering and 2) ineffectiveness when detecting parked domains containing external links that are usually abused by hackers, e.g., drive-by download attack. Aiming for alleviating above defects without sacrificing practical usability, this paper proposes *ParkedGuard* as an efficient and accurate parked domain detector. Several scripting behavioral features were analyzed, while those with special statistical significance are adopted in *ParkedGuard* to make feature engineering much more cost-efficient. On the other hand, finding memberships between external links and parked domains was modeled as a graph mining problem, and a coarse-to-fine strategy was elaborately designed by leverage the graphical locality such that *ParkedGuard* outperforms the state-of-the-art in terms of both recall and precision rates.

Keywords—Coarse-to-fine strategy, domain parking service, graphical locality analysis, parked domain.

I. Introduction

DUE to that a new type of advertising appearing on the web which is Pay-Per-Click (PPC) (also called as Cost-Per-Click (CPC)) in recent years, people, who own lots of registered domain name, realize that they could make better use of their domains rather than just waiting for someone buying them. Currently unused domain names could be lended to others who ”park”(put) their banners or PPC advertisings in these parked domains, such that the domain owners will get some remuneration as pay back. Therefore, the intermediate service that links domain owner and advisement syndicator helping them finding each other as well as setting up corresponding websites is called *domain parking service*.

However, when business scale of domain parking service grows larger and larger, it is unavoidable that some cyber-security-related issues will accompany as side effects,
such as parked domain monetization, abuses, and illicit activities [1]-[3], starting to influence internet users and causing lots of damage. Researchers were aware and put effort to detect parked domains to avoid users exposed to the threatens in the realm of domain parking services [4]-[7]. Among all the related works, for the purpose of early high-throughput screening, techniques tring to detect parked domain among millions of domain names using machine learning approaches [4] are the most major trend. Yet, to our best knowledge so far, the state-of-the-art detection method, Parking Sensors [4], suffers two primary defects. One is lacking efficiency as features generating in terms of CPU time to calculate and efforts to collect essential information. Besides, parked domains containing external links usually cannot be effectively detected by the previous work, whereas this kind of ”targeted parked domains” are most likely used to perform malicious activities, such as Drive-by Download attacks.

In this paper, to ameliorate the previous works defects without sacrificing practical usability, e.g., dropping the effectiveness, an efficient and accurate parked-domain detection system, named as *ParkedGuard* adopting coarse-to-fine strategy [8] was proposed as followings. At the first stage, to alleviate long latency of the aforementioned work, the proposed *ParkedGuard* considers several script behavorial indicators that describes the characteristics or the working semantics of domain web pages as features for parked domain prediction. Then, a statistical feature selection mechanism determines the most differentiable features according to their uni-variate significance. By means of significant feature subset selection from original feature set, *ParkedGuard* successfully reduces the CPU latency for calculating the values of indicators used in parked domain prediciton, as well as saves the efforts to collect essential information generating those inputs for feature engineering. These low-cost but high-descriptive feature subset was then used by random forest decision to pre-label our domain set which is early-stage decided as parked or non-parked domain candidates. On the other hand, the fine-tune part of proposed strategy is designed for not only compensating the resulted accuracy decreasing due to only a subset of original features was picked up, but also improving the prediction ability for important targeted parked domains. In the fine-tune process, one novel relational graph correlating both pre-labeled domain candidates and external links were constructed. And locality of each pre-labeled candidates was statistically analyzed and leveraged to re-check the predictions of other candidates around its neighborhood using nearest-neighbor based voting mechanism. In the final stage, the proposed *ParkedGuard* using only a few computational-efficient features produces significantly improved predictions to targeted parked domains, and comparable results for general cases, respectively.

Real-data evaluations show that 1) Comparing to using full features, the computational time needed for feature extracting decreases dramatically to only half of that used to be. In addition, efforts to collect essential information for generating full features could also be saved. 2) the proposed *ParkedGuard* considering graphical relationships between domains and their external links as well as locality among its neighborhood, successfully outputs precise predictions and especially emphasizes on targeted parked domains. The numerical performance matrices of *ParkedGuard* include accuracy, recall, precision, and f-measure are 90.4%, 93.1%, 85.4%, and 89.1%, respectively, which all outperform those metrics of previous work. Some real case studies demonstrated that *ParkedGuard* successfully identified several targeted parked domains whereas previous work failed to differentiate those targets from non-parked ones. As a result, when the scale of domains being checked becomes larger, our system not only maintains the low computation cost but also keeps the effectiveness about detecting parked domain.

II. Related Works

A. Domain Parking Service

Domain parking services refer to a registered domain name without being associated with any services such as a website or a mail server. The domain owners then temporarily ran it as an advertisement web portal to make a profit from the traffic the domain receives. In order to achieve this aim, the domain owner typically chooses to park the domain with a domain parking service, an intermediary between the owner and various monetization options. In a domain parking service, there are four important roles within it, the domain owner, the service provider, the advertisement syndicators, and the advertisers. The architecture of a domain parking service showed in Fig. 1.

Domain owners usually register an account with a domain parking service and park their domain on this service. After the domain being parked, parking service providers supply a site for domain owners to manage their parked domains [9], and they also cooperate with advertisement syndicators to provide advertisements on the parked page. The advertisement syndicators provide a platform for service providers and advertisers. They usually provide Javascript code to automatically generate advertisements on the parked page. So, the parked page will have advertisements to be clicked. However, this platform seldom detects the advertisement's content. Therefore, some malicious advertisers use this way to provide malicious advertisements on the parked page. This way might influence the users in the domain parking services.

B. Parked Domain Monetization Options

A domain parking service provides many monetization options to domain owners. The most popular ones are search

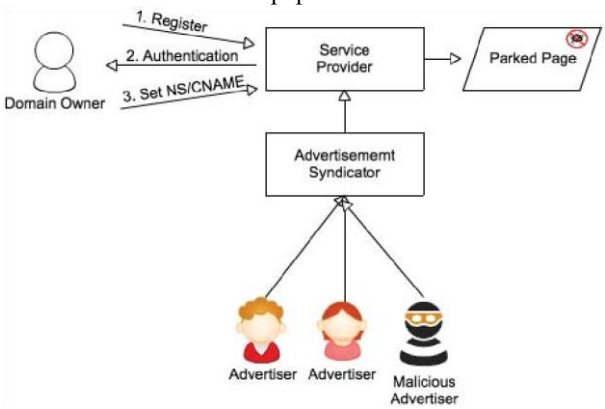


Fig. 1 The architecture of a domain parking service. The parked page would be automatically created by the service provider so the domain owner can easily earn the money

advertising, direct-navigation [10] monetization(Pay-Per Redirect). Search advertising possibly made through Pay-Per-Click (PPC) [11]-[13]. A parking service provider submits a search query for certain keywords (relevant to the domain names, in the case of parked domains) and receives relevant ads in the XML format, which also include the price per ad from the advertisers. The service providers then display a set of ads on the parked page. Once a user clicks on an ad, the click traffic is bounced through a number of hosts such as click servers before reaching the parked page. This click is paid for by the advertiser and the revenue generated in this way is shared between the domain owners, the service providers, and the ad networks. Another monetization method is direct navigation traffic, which is generated when the web user enters a domain name as a query and expects to be redirected to a related domain. For example, one may type in "findcheaphotels.com" in the address bar and land to mytravelguide.com. This is caused by a direct-navigation-traffic purchase that the owner of mytravelguide.com purchases through a direct navigation system the traffic related to keywords "travel" and/or "hotels." Parked domains can serve such a direct navigation system by redirecting type-in traffic to traffic buyers like mytravelguide.com. This monetization option is called Pay-Per-Redirect (PPR) or zeroclick [14], [15].

C. Illicit Activities in Domain Parking Services

Illicit activities in domain parking service can be discussed into two part. First part is the illicit monetization of parking service, such as click fraud, traffic spam, traffic stealing, Malware distribution, etc. Click fraud is a type of fraud that occurs on the Internet in pay-per-click (PPC) online advertising [16]-[18]. Fraud occurs when a person, automated script or computer program imitates a legitimate user of a web browser, clicking on the ads in order to earn money without having the actual interest in the target of the ad's link [19], [20]. A parking service may collaborate with traffic monetization platforms, which monetize different types of traffic such as parking traffic, error traffic (i.e. 404 not found pages) and non-existent domains [21], [22]. Alrwais et al. [5] observed that 70.7% of all PPR monetization are traffic spam in their experiments. They also found that some monetization chains going through their parked domains were not reported to domain owners but charged to their ad/traffic campaign accounts. Previous works studied that parked domain play a role within malware distribution [23], [6]. Alrwais et al. [5] also discovered that parking service involvement in malware distribution and they did not know.

Another part of illicit activities is the abuse of parking services, such as typo-squatting, trademark abuse, malicious redirections, malicious web infrastructures, etc. When a user type in long URLs, there is a possibility of a typing mistake and is not being detected. For example, typing and requesting wikipdeia.org instead of wikipedia.org, and started registering these typo-including domain names. As prior research has shown, the preferred monetization strategy of typosquatters is parked domains [3], [24], [25]. Typosquatting domains is in the trademark abuse category, but it's not saying that every domain which abuses trademarks is certainly a typosquatting domain [4], [26], [27]. For example, the currently parked domain facebookonline.com. This domain abuses Facebook's trademark, but it would never be automatically generated by the typosquatting models that mentioned before [5]. Such abusive domains have often been associated with phishing since they are so similar to the real trademark domain. Those illicit activities not only hurt the victims visiting a parked domain but also affect the parked domain when it gets blacklisted by URL scanners,

which reduced the value when the domain owner decide to sell it.

D. Parked Domain Analysis/Detection and Its Current Problem

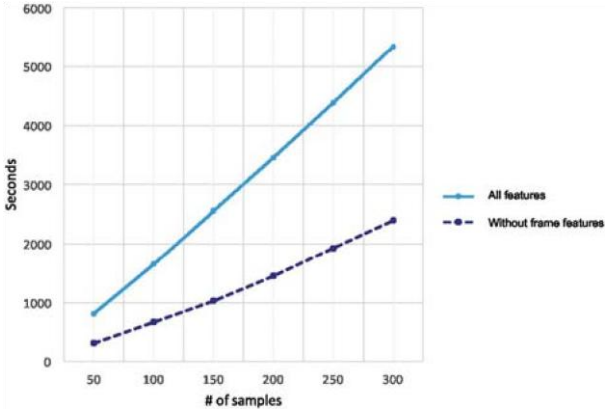
In 2010, Almishari et al. [28] developed a classifier for "ads-portal" domains, which they used to identify typosquatting abuse of domain parking services. They observed that in 2008, 50% of parked pages were residing on typosquatting domains. The classifier they developed leveraged several HTML features. But they did not use the features that logging the interaction between a web browser and a site, i.e., HAR features. Moreover, the authors did not consider about the site redirections and the frame item in a web [4]. Alrwais et al. [5] registered themselves both as domain owners and advertisers, effectively operating at both ends of the domain parking ecosystem. The authors investigated the domain parking industry, focusing on the fraudulent practices in their monetization chain. In 2014, Kuhrer et al. [29] evaluating the effectiveness of malware blacklists. In their evaluation step, they proposed a mechanism to identify parked domains by training an SVM classifier on seven inherent features they identified for parked web sites, such as HTTP redirection behavior, the normalized Levenshtein ratio [30] between the HTML content and the ratio of human-readable text in relative to the overall length in returned web content after removing HTML tags, JavaScript codes, and whitespaces. Using these proposed methods to evaluate whether the blacklists have the parked domains or not. They have shown that many parking providers reuse popular malware domains. In 2015, Vissers et al. [4] provided a client-side countermeasure to protect the user-part of this ecosystem. They designed and built a parked-page classifier which can be used to, block parked pages or alert users that they are currently interacting with a parked page. They propose four categories of features: HTML features, HAR features, frame features, and domain name features. The performance of their work is a 99.65% area under the curve (AUC). However, they did not consider the targeted parked domain, a parked domain that focuses on some ads and without significant scripting behavior characteristics of parked domain but still can be observed by their relationships of links. Our works focus on this kind of parked domain and show that the targeted parked domain still can be discerned by their relationships of links.

III. Method and System Architecture

Section III explained the detailed methodology and system architecture of *ParkedGuard* as shown in Fig. 2. *ParkedGuard* is designed to operate in an enterprise network space. It can not only detect parked domain but help us to increase the speed of detecting malicious domain as well. Owing to previous works have high latency to detect parked domain and cannot detect the domains without significant characteristics of abuse and malicious activities, also called *targeted parked domain*, we propose an efficient methodology to solve the latency problem and discover targeted parked domain.

A. Scripting Behavior Features Extracting

Scripting behavior features categorized into HTML, HTTP Archive, frame, and domain name features [4] were the characteristics of domain web page, totally 22 features. The HTML features are the features that extracted from the source code of every loaded frame, the HAR features are the features derived from the HTTP Archive (HAR) that is created while the website is loading, the frame features are the features that tracking every loaded frame on the web page and the domain name features are the features that focus on characteristics inferred from the domain name itself. It had the problem of high computational time about the feature extraction, especially the type of frame features. After using Parking Sensor, our opinion was that it had the time-cost problem in the feature-extracting phase making it inefficient to operate in a large-scale detection. The observation after diving into this problem is that some features have high latency but low efficiency. Two analyses were discussed in the bellowing list. • The efficiency about calculating the high latency features: This experiment analyzed the efficiency about calculating the high latency features. It took off the high latency features from the Parking Sensors and recorded the features extracting time with high latency features and without high latency features. The result has shown in Fig. 3 that the calculation time about features decreases to half of the time when using the n = 300 samples. With the number of n being larger, we can save more time in feature extraction.



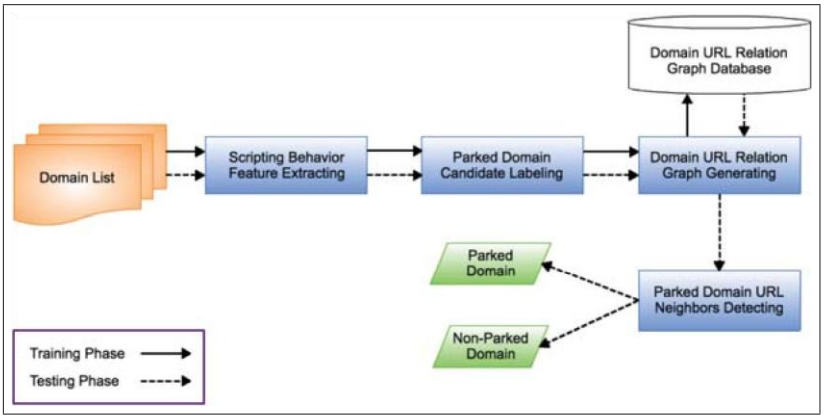


Fig. 2 The system architecture of ParkedGuard. Use domain list as input, and produces domain URL relation graph to find the parked domain as output

Fig. 3 Time cost of feature extracting for different feature sets

TABLE I
THE RESULT OF PARKING SENSORS WITH TWO DIFFERENT FEATURE SETS

	All features	Without high latency features
Precision	85.7%	80.6%
Recall	83.0%	81.8%

. The effectiveness about the high latency features: This experiment which is designed to detect parking domains with all features, Parking Sensor provide, and with just low latency features verified whether the high latency features are important in the Parking Sensors. It ran in Orange3 which is a component-based data mining software [31], then executed with 300 samples implemented with 10 cross-validation. The result is in Table I. It showed that the precision and recall between the feature set without frame features and the feature set with all features were very nearly.

After analyzing the sample set for the feature extraction time and the importance of having the high latency features in detection, our conclusion was to select a feature set from Parking Sensors that can not only decrease much extraction time but also maintain the effectiveness of detection. It came into Parked domain scripting behavior feature extracting module for selecting features which have the characteristics of domain web page as we shown in Fig. 4 The input is a domain list, and the output is a set of features. First, high latency features were excluded. Second, it executed with 2000 samples implemented 10 cross-validation in the experiment, and randomly select 50 feature sets from 50 samples. Last, as the results of each feature were not a normal distribution, Kruskal-Wallis test [32] is considered as the method of our feature selection. The Kruskal-Wallis test is a non-parametric method for testing whether samples originate from the same distribution. The results of Kuskal-Wallis test is presented in Table II. 8 features were selected from 22 features, Parking Sensors proposed by judging with Kuskal-Wallis test where their p-value are smaller than 0.05.

Fig. 4 Scripting behavior feature extracting: the input is a domain list, and the output is a set of features

TABLE II
THE RESULT OF KRUSKAL-WALLIS TEST ABOUT THE FEATURE WITHOUT FRAME FEATURES

Feature Name	Kruskal-Wallis Statistic	P-value
Amount of Meta Refreshes	21.711	
Link-to-Global Text Ratio	11.449	0.001
Amount of Non-Link Characters	10.905	0.001
Amount of Window Location	9.646	0.002
Average Source Length	9.041	0.003
Text-to-HTML Ratio	9.139	0.003
External Source Ratio	8.004	0.005
Maximum Link Length	6.271	0.012

The selected features were average source length, external source ratio, link-to-global text ratio, maximum link length, amount of meta refreshes, amount of non-link characters, text-to-HTML ratio and amount of window location. Detailed explanations list in the fellowing paragraphs:

- Amount of Meta Refreshes

  Parked domains usually use redirection mechanisms to lead visitors to other pages or domains. Meta Refreshes is one of a redirection mechanisms in HTML. Although non-parked domains might have redirection mechanisms, it still can assist the classification when combining other features. It can be calculated by looking for http-equiv="refresh" in the HTML files to extract this feature.

- Amount of Window Location

  Window location is another redirection mechanism of JavaScript redirection code. It can be calculated by searching for window.location in the HTML file to calculate the amount of the presence words.

- Link-to-Global Text Ratio

  Many parked domains have less text that is not the part of the links. On a typical parked page, text is either part of an

ad or part of the ”Related Links.” To this reason, this feature extracts all text from the HTML pages with Pythons Natural Language Toolkit [33], which return the text without HTML tags. Then, it calculated the ratio of the amount of text that in the links (i.e. <a> element and its child nodes) and the global amount of text display on the page.

- Amount of Non-Link Characters

  The feature counts the actual amount of characters not belonging to any link element, instead of exclusively relying on the ratio.
- Maximum Link Length

  Owing to the advertisement links, which are the major component on parked domain web pages, pass more and longer parameters along with the link in order to track the click on the PPC ad. This feature count the number of <a> elements display on the page and measure the string length of the destination addresses. Using these numbers, we can calculate the maximum link length of the page.
- Average Source Length

  Similar to the previous feature, source addresses for banners and other advertisement media, tend to pass parameters of campaigns, image dimensions, etc. The non-parked web pages were expected to have more static media sources and thus shorter address lengths.
- Text-to-HTML Ratio

  It can be measured by the ratio of text to the total amount of characters in the HTML file. This feature focuses more on the dynamic generation of content.
- External Source Ratio

  An external source can be defined as one with an address pointing to another domain. Links and media generated by third-party advertisement syndicators will usually display on domains of that syndicator. The non-parked web pages were expected to have a lower ratio of external links because they commonly also have links to pages and media hosted on the same domain.

Fig. 5 Parked Domain Candidate Labeling: the input is a set of features, and the output is pre-labeled candidate domain list

Fig. 6 Domain URL Relation Graph Generation: the input is pre-labeled candidate domain list and the output is a domain URL relation graph

B. Parked Domain Candidate Labeling

In this module, it take the output of previous module, a set of feature vector, as input shown in Fig. 5. Due to aiming for high interpretability of our learning algorithm, it is important to comprehend the prediction of learning algorithm of the module for further improvement and adaptability. Therefore, Random forest algorithm was implemented, as it combines the strength of the ensemble learning with the interpretable qualities of decision trees. Moreover, it also improves the decision tree that tends to be robust with regard to outliers, the ensemble method of Random Forest protects the model against overfitting. Besides, after the tree have been constructed in the learning phase, the classification is usually very quick in the predicting phase. After the ensemble learning is done, we put the candidate domain in the decision tree to help us label it. Finally, we get the candidate domain that can be pre-labeled as parked or non-parked.

After the ensemble learning is done, we put the candidate domain in the decision tree to help us label it. Finally, we get the candidate domain that can be pre-labeled as parked or non-parked.

C. Domain URL Relation Graph Generation

The input in this module is pre-labeled candidate domain list and the output is a domain URL relation graph as shown in Fig. 6. To compensate for efficiency of computational time trade-off, *ParkedGuard* adopt graphical locality analysis, constructing the domain URL relation graph G for each candidate domain, as shown in Fig. 7. A domain URL relation graph is defined as below:

Let D is a set of candidate domain shown in (1) and E is a set of external URL links of D_m shown in (2),

$$D = \{D_m \mid D_m \text{ is a candidate domain}\} \quad (1)$$

$$L_m = \{L^n_m \mid L^n_m \text{ is an external URL link of } D_m\} \quad (2)$$

Let $G = (V,E)$ is a domain URL relation Graph where V is a set of vertices shown in (3) and E is a set of edges shown in (4),

$$V = \{D, L_m\} \quad (3)$$

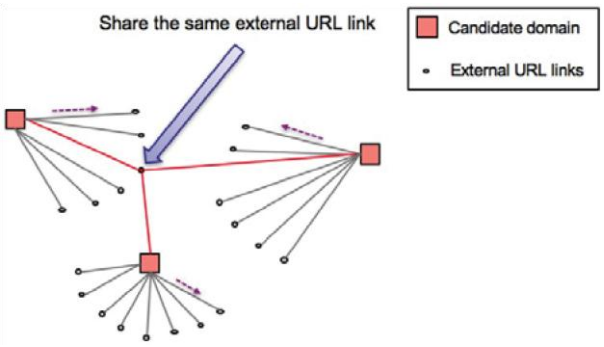


Fig. 7 The schematic diagram of domain URL graph of candidate domains

$$E = \{E_i \mid E_i \text{ is the relationship between } D_i \text{ and } L_m\} \quad (4)$$

The external URL link is defined as a hyperlink that point at any second level domain other than the second level domain

where the link exists, as shown in Fig. 8.Finally, we combine all domain URL relation graphs for each candidate domain D_m and then get the last graph showed in Fig. 7. This figure demonstrates that different domains might share the same external URL link. We use the output from the Parked Domain Candidate Labeling in Section III-B (i.e. pre-labeled candidate domain) to draw the graph in Fig. 10. The node of false positive is the targeted parked domain which without significant scripting behavior characteristics of the parked domain but still can be observed by their relationships with other domains. We use the relationships in the domain URL relation graph shown in the following to detect the parked domains, especially the targeted parked domains.

D. Parked Domain URL Neighbors Detection

In this module, the input is the domain URL relation graph which output from the Domain URL Relation Graph Generation in Section III-C as shown in Fig. 9. We proposed an algorithm called *Parked Domain URL Neighbors Detection* as shown in Algorithm 1. The algorithm uses the relationship between each D_m and L^n_m to label the parked domain as the following steps:

1) Labeling the isolated domain that without any domain neighbors by the output label in Section III-B.
2) Sorting each L^n_m by their degrees from high to low due to the concept that the high degree external URL link has more confidence in which its neighbors are the same class of domain. Each L^n_m would have a set of D_m.
3) Sorting each D_m in the set of each L^n_m by the degree of D_m from low to high due to the previous concept that high degree L^n_m have the same class of domain. That is, if a D_m only has a neighbor that is a high confidence L^n_m, then the D_m also be high confidence.

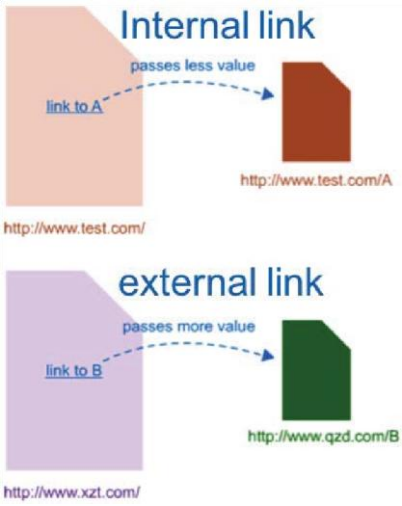


Fig. 8 The definition of internal URL link and external URL link

Fig. 9 Parked Domain URL Neighbors Detection: the input is the domain URL relation graph and the output is that a candidate domain is parked or not

4) Using 2, 3 to traverse the domain URL relation graph, and calculating the two scores of each D_m, parked domain score and non-parked domain score. In this part, we calculate the score of each D_m's one stage neighbor L^n_m. If L^n_m's one stage neighbor exclude D_m is a parked domain, then add one point to the parked domain score.

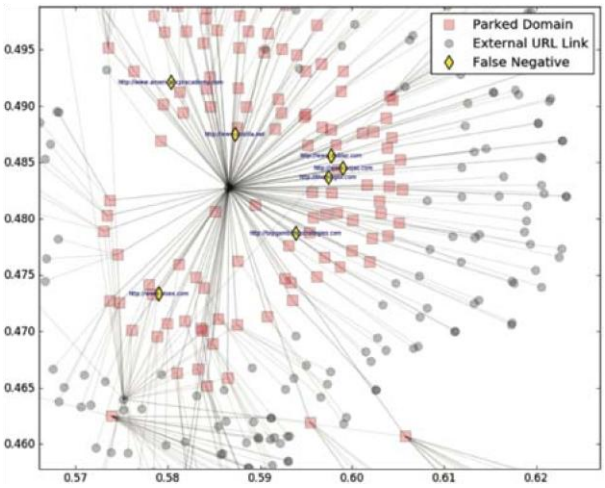


Fig. 10 The parked domain URL relation graph. The square is the parked domain. The circle is the external URL links. The diamond is the false negative of the previous work, and it is the targeted parked domain which we want to detect using its relationship with other domains in the graph

On the contrary, if L^n_m's one stage neighbor exclude D_m is a non-parked domain, then add one point to the non-parked domain score.

5) Labeling D_m by comparing the parked domain score and non-parked domain score which is higher.

Algorithm 1 Parked Domain URL Neighbors Detection

Require: $G = (V,E)$

$DLV = \{\{D, LD_m|\ D\}_m$ *is a candidate domain*$\}$

Ensure:

$m = \{L^n_m \mid L^n_m \text{ is an external URL link of } D_m\}$

$D = \{D_m \ (D_m = Parked\ Domain) \vee (D_m = Non-Parked\ Domain)\}$

1:2: $D_m.degree() = 0$ **then** forif*all* $D_m \in G.node()$ do
$D_m = label\ of\ D_m$
3:
4: **end if**
5: **end for**
6: $SortedL = a\ list\ of\ L_m^n\ sorted\ by\ degree\ from\ high\ to\ low$
7: **for** *all i in SortedL* **do**
8: $SortedD = a\ list\ of\ i's\ neighbor\ D_m\ sorted\ by\ degree\ from\ low\ to\ high$
9: $ScorePD = 0$
10: $ScoreNPD = 0$
for *all j in SortedD* **do**
11: $OneStageNeighborOfj$ $a\ list\ of\ j'sone\ stage\ neighbor$
12: =
13: for *all k in OneStageNeighborOfj* do
14: *OneStageNeighborOfk*
= *'a list of k s one stage neighbor stage neighbor*
15: for *all l in OneStageNeighborOfk* do
16: if *l = Parked Domain* then
17: *ScorePD = ScorePD* +1
18: else
19: *ScoreNPD = ScoreNPD* +1
20: end if
21: end for
22: end for
23: if *ScorePD > ScoreNPD* then
24: *Dm = Parked Domain*
25: else
26: $D_m = Non - Parked\ Domain$ end if
27:
28: end for
29: end for

IV. Experiments and Results

The goal of experiments intends to measure the effectiveness of detecting parked domains against the abuse of the domain parking service ecosystem in real world. They are designed for 1) the efficiency of *ParkedGuard* which deliberately get rid of the high execution time features and 2) the effectiveness of *ParkedGuard* using graphic locality to compensate for the removed features.

A. Description of Dataset

The dataset of our experiment was collected by searching the records of the DNS Census dataset [34], which contains about 2.5 billion DNS records gathered in 2012 and 2013. All domains matched the DNS configurations listed in Table III which Thomas et al [4]. collected are extracted then queried domain's DNS records to confirm whether they were still parked with that particular parking service. There are totally 11,406,099 parked domains from 15 observed parking services and 223,705,447 normal domains. Since the DNS census is outdated, this indicates that at the time of this writing there exist at least 11 million domains whose is to serve ads while they are visited.

TABLE III
The Parking Services with Their Required Domain

No	Parking Service	Setting	Address
	SedoParking	NS	sedoparking.com
	InternetTraffic	NS	internettraffic.com
	CashParking	NS	cashparking.com
	Fabulous	NS	fabulous.com
	DomainSponsor	NS	dsredirection.com
	Above	NS	above.com
	ParkingCrew	NS	parkingcrew.net
		A	62.116.181.25
		CNAME	parkingcrew.net
	Skenzo	NS	ztomy.com
	NameDrive	NS	fastpark.net
	Voodoo	NS	voodoo.com
	RookMedia	NS	rookdns.com
	Bodis	NS	bodis.com
		CNAME	parking.bodis.com
	DomainApps	NS	domainapps.com
	TrafficZ	NS	trafficz.com
		A	198.202.142.246
		A	198.202.143.246
	TheParkingPlace	NS	pql.net
		CNAME	putoppose.net/d/domain

TABLE IV The Dataset Description

	Number of Domain	Duration
Parked Domain		2012-2013
Non-Parked Domain		2012-2013
Total		

Training and testing data. We random select 2000 parked domain and 2000 non-parked domain as our training data. Next, we also random select 921 parked domain and 664 non-parked domain as our testing data. Table IV shows the description of our dataset.

B. Evaluation Metrics

ParkedGuard regards each domain as an instance and identifies parked domains as positive instances whereas non-parked domains as negative. Confusion matrix and its derivations provided a set of deliberately metrics in the following evaluation. TP (true positive) is the number of parked domains correctly detected; FN (false negative) is the number of parked domain misclassified as normal ones. TN (true negative) is the number of the non-parked domains that are correctly classified; FP (false positive) is the number of the non-parked domains that are wrongly classified as parked domains. Fellowing description illustrates the meaning of derivations from confusion matrix.

• The *Accuracy* is defined by

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (5)$$

• The *Recall (True Positive) rate* is defined by

$$\text{Recall rate} = \frac{TP}{TP+FN} \quad (6)$$

• The *Precision* is defined by

$$\text{Precision} = \frac{TP}{TP+FP} \quad (7)$$

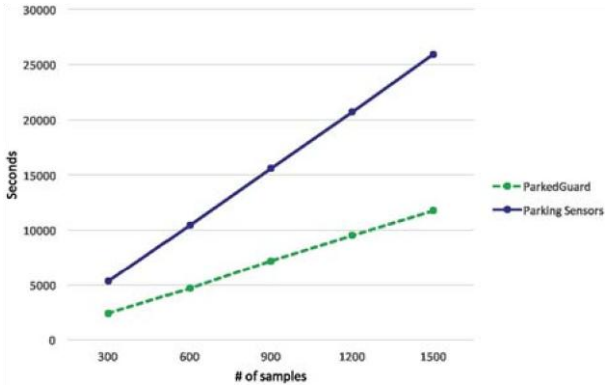


Fig. 11 The relationship between the execution time and the number of samples

TABLE V
THE CONFUSION MATRIX OF PARKING SENSORS AND *ParkedGuard* (a)
THE CONFUSION MATRIX OF PARKING SENSORS Predicted as -> Parked

	Non-Parked	Total	
Parked	610	54	664
Non-Parked	113	805	921
Total	727	858	1585

(b) THE CONFUSION MATRIX OF PARKEDGUARD

Predicted as -> Parked	Non-Parked	Total
Parked	618	46	664
Non-Parked	106	815	921
Total	724	861	1585

• The *F-measure* is defined by

$$\text{F-measure} = \frac{2 \times Recall \times Precision}{Recall + Precision} \quad (8)$$

C. The Efficiency of ParkedGuard

Since *ParkedGuard* only chose the essential 8 features from Parking Sensor proposed and appended graphic locality analysis, It is necessary to have evidence that how much computation time *ParkedGuard* are capable of saving. For the efficiency analysis, the experiment revealed the compared execution time of *ParkedGuard* with Parking Sensor. All of the experiments are done in an environment with CPU: 2.4 GHz Intel Core i5 Memory: 8 GB 1600 MHz DDR3 Graphics: Intel Iris 1536 MB OS: OSX El Capitan Version 10.11.5. *ParkedGuard* and Parking Sensors were separately running on testing data, depicted in IV-A, where quantities of samples were put into from 300, 600, 900, 1200 and 1500 sequentially. The proportion of parked domains and non-parked domains are equal, randomly picking up from the training dataset.

The results shown in Fig. 11 express that the execution time of *ParkedGuard* was roughly half of the one of Parking Sensor. As the quantities of domains needed to detect increase, *ParkedGuard* saves much more time to execute the detection process.

D. The Effectiveness of PackedGuard

Previous result of experiments IV-C proved that *ParkedGuard* is time-saving detector in comparison with Parking Sensor. The experiments in this section are about to

TABLE VI THE COMPARISON OF EVALUATION METRICS

Target	Accuracy	Recall	Precision	F-measure
ParkedGuard	0.904	0.931	0.854	0.891
Parking Sensors	0.893	0.918	0.844	0.879

investigate the detecting effectiveness of these two detectors. Both of them initially trained their own detection model with training data, then started to detect with testing data. Table V shows two confusion matrices of experiments setting for Parking Sensor and *ParkedGuard* respectively. According to both confusion matrices, four reliable metrics mentioned in Section IV-B express more understandable comparison. From Table VI, all the four metrics of *ParkedGuard* beats the equivalents of Parking Sensor separately. To compensate for the efficiency of computational time trade-off, *ParkedGuard* adopt graphical locality analysis hence it even improve the effectiveness of detection compared with Parking Sensor.

E. Case Studies

This subsection tells two different cases that on one is the targeted parked domain which *ParkedGuard* can detect and another one is the isolated parked domain that *ParkedGuard* miss.

- The targeted parked domain which without significant scripting behavior characteristics of parked domain could be observed by their relationships of links as *ParkedGuard* adopted the relationship between domain and external URL links to find the targeted parked domain as we shown in Fig. 12.
- Even *ParkedGuard* taking graphic locality into account, the domain which is isolated and has no relationship with each other, like http://sexlinkje.com in Fig. 13, was misclassified as a normal one.

F. Experiment Discussion

Previous work, Parking Sensor, can only detect parked domain but it could not identify targeted parked domain that without significant scripting behavior characteristics of parked domain which can be observed by their relationships of links in *ParkedGuard*The evaluation results show that˙ *ParkedGuard* can detect the parked domain especially targeted parked domain using the relationship between domain and its external URL links. In Table VII, it presents the comparison between previous work and *ParkedGuard*˙*ParkedGuard* using low latency features thus the execution time is lower than previous work. The

experiment results of *ParkedGuard* show that Accuracy can achieve 90.4%, Recall is 93.1%, Precision is 85.4%, and F-measure is 89.1%. Therefore, when the number of samples becomes larger, our system can not only cost down the time but also maintain the effectiveness about detecting parked domain.

V. Conclusion

This paper proposes *ParkedGuard*, a combination of signature-based mechanism and domain URL relation graph

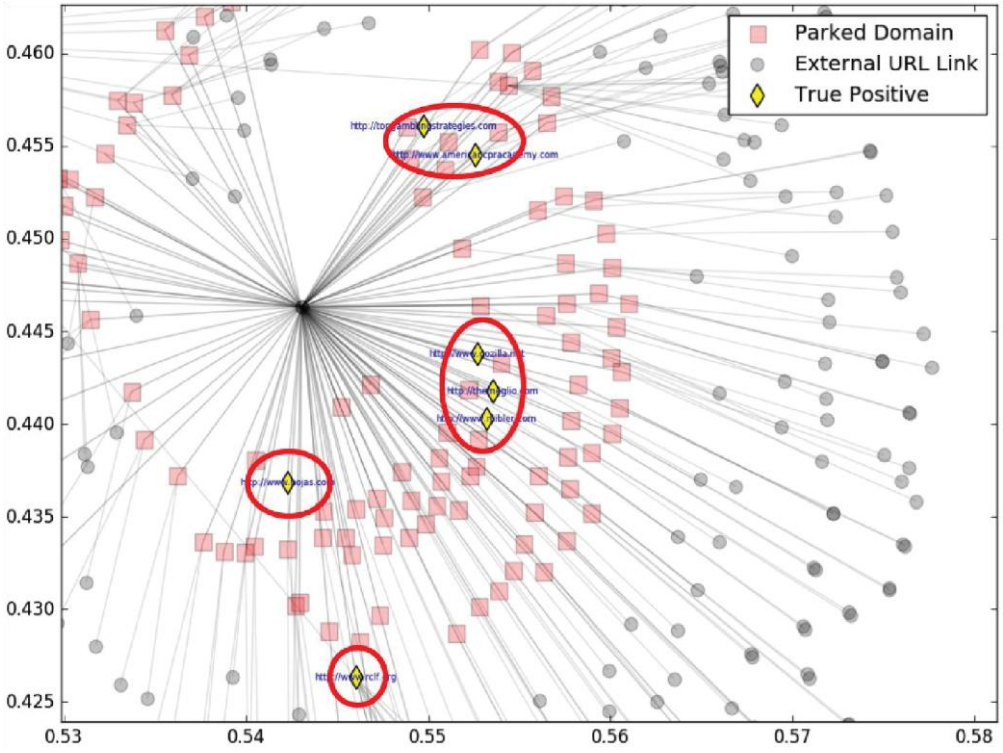


Fig. 12 The diamond is the targeted parked domain which ParkedGuard can detect

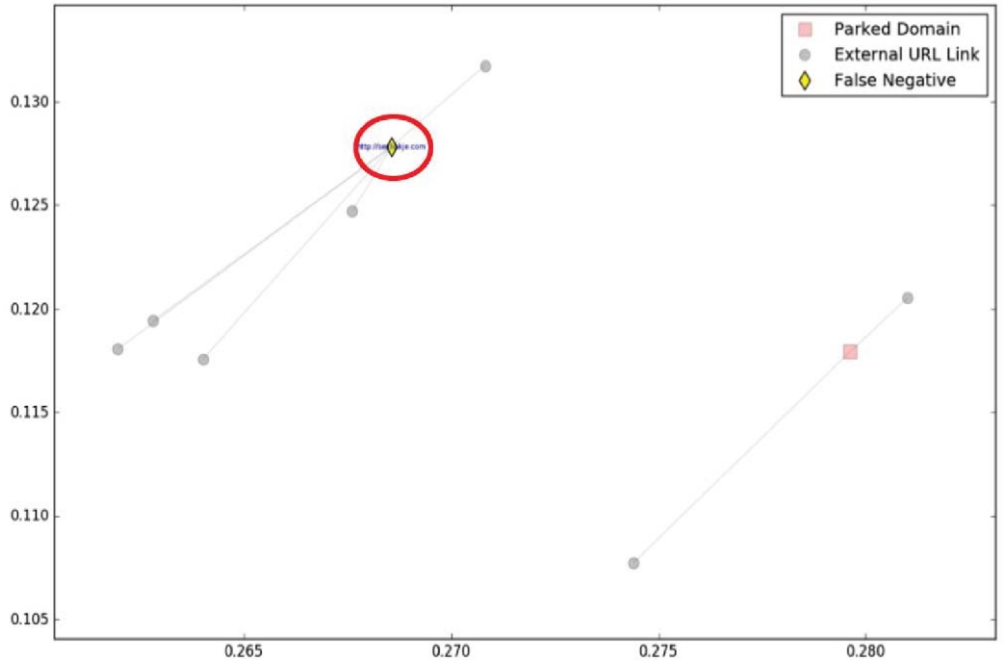


TABLE VII
THE COMPARISON TABLE OF *ParkedGuard* AND PARKING SENSORS [4]

approach to detect the parked domain, especially the ”targeted parked domain” which without significant scripting behavior characteristics of parked domain but still can be observed by their relationships of links.

ParkedGuard has the following properties. Effectiveness: It is effective, that is, it is able to distinguish parked domain and non-parked domain in the network. Scalability: It achieves up to 90.4% percentage points in accuracy. It is scalable, that is, it is linear in the size of the problem (i.e., the number of domains in the input list). Efficiency: It does not need to generate too many features, especially the long latency features: frame features and have higher accuracy than the Parking Sensors that need to calculate more features.

Furthermore, *ParkedGuard* also makes the following contributions:

- Improving the high latency problem in calculating scripting behavior features.
- Proposing a Domain URL Relation Graph Generator to detect targeted parked domain.
- Proposing an algorithm called Parked Domain URL Neighbors Detection to detect parked domain, especially the targeted parked domain.
- Developing a system to detect the parked domain, especially the targeted parked domain.

Although this paper focused on the parked domain, especially the targeted parked domain. *ParkedGuard*combined the signature-based behavior features and the relationship between the domain and its external URL links in the relation graph. Because some domain has no relationship with others, it made graphic locality ineffective. Therefore, with more efficient perspective, the next step we need to survey is to define the isolated domain on the domain URL relation graph.

REFERENCES

[1] D. Kesmodel, *The Domain Game: How People Get Rich from Internet Domain Names*. Xlibris Corporation, 2008.

[2] P. Agten, W. Joosen, F. Piessens, and N. Nikiforakis, “Seven months’ worth of mistakes: A longitudinal study of typosquatting abuse,” in *Proceedings of the 22nd Network and Distributed System Security Symposium (NDSS 2015)*. Internet Society, 2015.

[3] T. Vissers, W. Joosen, and N. Nikiforakis, “Parking sensors: Analyzing and detecting parked domains,” in *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS ’15)*, 2015.

[4] S. Alrwais, K. Yuan, E. Alowaisheq, Z. Li, and X. Wang, “Understanding the dark side of domain parking,” in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 207–222.

[5] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang, “Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures,” in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 112–126.

[6] L. Metcalf and J. Spring, “Domain parking: Not as malicious as expected,” DTIC Document, Tech. Rep., 2014.

[7] Y. Amit, D. Geman, and X. Fan, “A coarse-to-fine strategy for multiclass shape detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 12, pp. 1606–1621, 2004.

[8] Sedo,“Domain parking terms and conditions.” accessed on

[9] K. Hartog, “System and method for pay-per-click revenue sharing,” Mar. 22 2005, uS Patent App. 11/086,813.

[10] L. Zhang and Y. Guan, “Detecting click fraud in pay-per-click streams of online advertising networks,” in *Distributed Computing Systems, 2008. ICDCS’08. The 28th International Conference on*. IEEE, 2008, pp. 77–84.

[11] T. P. Barber, “Method of charging for pay-per-access information over a network,” Jul. 27 1999, uS Patent 5,930,777.

[12] ——, “Bandwidth-preserving method of charging for pay-per-access information on a network,” Dec. 5 2000, uS Patent 6,157,917.

[13] S. A. Alrwais, A. Gerber, C. W. Dunn, O. Spatscheck, M. Gupta, and E. Osterweil, “Dissecting ghost clicks: Ad fraud via misdirected human clicks,” in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 21–30.

[14] T. Blizard and N. Livic, “Click-fraud monetizing malware: A survey and case study,” in *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*. IEEE, 2012, pp. 67–72.

[15] B. Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson, “Whats clicking what? techniques and innovations of todays clickbots,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2011, pp. 164–183.

[16] V. Dave, S. Guha, and Y. Zhang, “Viceroi: Catching click-spam in search ad networks,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 765–776.

[17] P. Pearce, C. Grier, V. Paxson, V. Dave, D. McCoy, G. M. Voelker, and S. Savage, “The zeroaccess auto-clicking and search-hijacking click fraud modules,” DTIC Document, Tech. Rep., 2013.

[18] V. Dave, S. Guha, and Y. Zhang, “Measuring and fingerprinting click-spam in ad networks,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 175–186, 2012.

[19] J. Jung and E. Sit, “An empirical study of spam traffic and the use of dns black lists,” in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 370–375.

[20] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, “Measuring pay-per-install: The commoditization of malware distribution.” in *Usenix security symposium*, 2011, p. 15.

[21] J. Szurdi, B. Kocso, G. Cseh, J. Spring, M. Felegyhazi, and C. Kanich, “The long taile of typosquatting domain names,” in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 191–206.

[22] Y.-M. Wang, D. Beck, J. Wang, C. Verbowski, and B. Daniels, “Strider typo-patrol: Discovery and analysis of systematic typo-squatting.” *SRUTI*, vol. 6, pp. 31–36, 2006.

[23] R. Bhalla, “Trademark trafficking in cyberspace an analytical study,” 2011.

N. Nikiforakis, S. Van Acker, W. Meert, L. Desmet, F. Piessens, and W. Joosen, “Bitsquatting: Exploiting bit-flips for fun, or profit?” in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 989–998.

[24] M. Almishari and X. Yang, “Ads-portal domains: Identification and measurements,” *ACM Transactions on the Web (TWEB)*, vol. 4, no. 2, p. 4, 2010.

[25] M. Kuhrer, C. Rossow, and T. Holz, “Paint it black: Evaluating the¨ effectiveness of malware blacklists,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 1–21.

[26] F. J. Damerau, “A technique for computer detection and correction of spelling errors,” *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[27] “Orange3,” accessed on 2017-02-28. (Online). Available: https: //github.com/biolab/orange3

[28] E. Theodorsson-Norheim, “Kruskal-wallis test: Basic computer program to perform nonparametric one-way analysis of variance and multiple comparisons on ranks of several independent samples,” *Computer methods and programs in biomedicine*, vol. 23, no. 1, pp. 57–62, 1986.

[29] S. Bird, “Nltk: the natural language toolkit,” in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.