Cracking Classifiers for Evasion: A Case Study on the Google's Phishing Pages Filter

Bin Liang, Miaoqiang Su, Wei You, Wenchang Shi, Gang Yang
Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), MOE, Beijing 100872, P. R. China
School of Information, Renmin University of China, Beijing 100872, P. R. China
{liangb, sumiaoqiang, youwei, wenchang, yanggang}@ruc.edu.cn

ABSTRACT

Various classifiers based on the machine learning techniques have been widely used in security applications. Meanwhile, they also became an attack target of adversaries. Many existing studies have paid much attention to the evasion attacks on the online classifiers and discussed defensive methods. However, the security of the classifiers deployed in the client environment has not got the attention it deserves. Besides, earlier studies only concentrated on the experimental classifiers developed for research purposes only. The security of widely-used commercial classifiers still remains unclear. In this paper, we use the *Google's phishing pages filter* (GPPF), a classifier deployed in the *Chrome* browser which owns over one billion users, as a case to investigate the security challenges for the client-side classifiers. We present a new attack methodology targeting on client-side classifiers, called *classifiers cracking*. With the methodology, we successfully cracked the classification model of GPPF and extracted sufficient knowledge can be exploited for evasion attacks, including the classification algorithm, scoring rules and features, etc. Most importantly, we completely reverse engineered 84.8% scoring rules, covering most of high-weighted rules. Based on the cracked information, we performed two kinds of evasion attacks to GPPF, using 100 real phishing pages for the evaluation purpose. The experiments show that all the phishing pages (100%) can be easily manipulated to bypass the detection of GPPF. Our study demonstrates that the existing client-side classifiers are very vulnerable to classifiers cracking attacks.

CCS Concepts

• Security and privacy→Browser security • Security and privacy→Phishing • Social and professional topics→Software reverse engineering • Computing methodologies→Machine learning.

General Terms

Security

Keywords

Phishing Detection; Machine Learning; Classifiers; Cracking; Collision Attacks; Evasion Attacks

WWW 2016, April 11-15, 2016, Montréal, Québec, Canada.

ACM 978-1-4503-4143-1/16/04.

http://dx.doi.org/10.1145/2872427.2883060

1. INTRODUCTION

Machine learning techniques have been commonly adopted in security applications. Various classifiers were trained for detecting malicious web pages [25], spam [54], phishing [58], malware [47], etc. Not surprisingly, the classifiers themselves have also become an attack target of adversaries. The adversary can attempt to fool classifiers by purposely modifying their behaviors. For example, a spammer can manipulate the spam mails to evade spam filters by inserting some *good* words indicative of legitimate mails or misspelling *bad* words indicative of spam mails [39]. This requires the classifier to be able to resist potential attacks.

Many existing studies have paid attention to the security of classifiers. According to the taxonomy of attacks against classifiers [9][10][32], the influences of attacks on the classifier are categorized into two types: (1) *causative* attacks interfere training process with control over the training data to downgrade the performance of the classifier, and (2) *exploratory* attacks exploit the knowledge of the trained classifier to cause misclassifications without affecting the training process.

In causative attacks, the adversary has the opportunity to inject (poison) specially crafted samples during the collection of training samples. Such attacks will cause the learner to misclassify security violations (i.e., false negatives) [17][18][19][20]. For example, a poisoning attack method against support vector machines (SVM) is presented in [18]. It was demonstrated that the SVM's classification accuracy can be largely impacted by feeding malicious training data. Fortunately, in practice, the adversary doesn't always have an opportunity to effectively control over training data. In fact, the training process of most classifiers, especially the ones deployed in commercial products, is not open to the public. The adversary needs to fight with trained classifiers. For example, the Google's server-side phishing page classifier is developed in an offline training process [58], whose training dataset consists of millions of samples from various domains. In this case, it is very difficult, if not impossible, for an adversary to craft sufficient amounts of malicious inputs to effectively poison the training process.

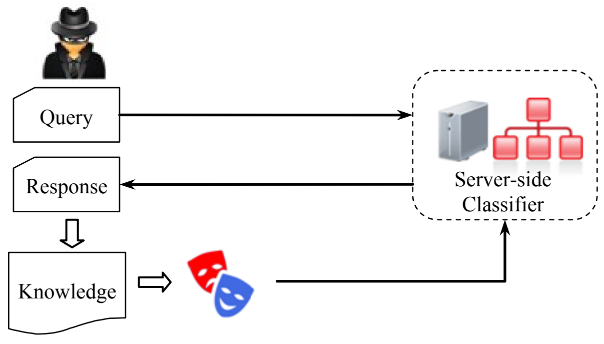
On the other hand, exploratory attacks attempt to learn enough knowledge about the trained classifiers and to find a way to evade the classification. Some existing studies on evasion attacks made the unrealistic assumption that the adversary has perfect knowledge of classification model [28]. In practice, the adversary often needs to send some probes (e.g., membership queries) to the classifier and then observe its response so as to deduce desirable knowledge [25], perform an adversarial learning to get sufficient knowledge about the target classifier to construct evasion attacks [40], or reconstruct an imitation of the target classifier based on the available public information (e.g., training data) to gain key knowledge [51]. In theory, the success of evasion attacks heavily depends on the amount of knowledge possessed by the adversary. Especially, the knowledge about features contributes most to the success of the attacks as discussed in [52]. Accordingly, some mitigation techniques have been proposed to against evasion attacks by either reducing the leakage of exploitable knowledge as much as possible [9][11] or making the learning method more robust to evasion [16][36].

However, existing studies often overlooked an important fact that some classifiers are deployed in the client environment that is fully controlled by users (*client-side classifier* for short) rather than in a remote server. For example, the classifiers for filtering spam emails and phishing pages are often embedded in the email clients or web browsers respectively. In the scenario, the classifiers face more serious security challenges. Instead of collecting the information via indirectly observations, the adversaries can freely and directly analyze the implementation and configuration of the classifiers to evade them. Consequently, it should be investigated carefully that how an adversary can learn the exploitable knowledge from a classifier deployed in the user clients and how effectively the knowledge are exploited in launching an evasion attack. Additionally, the existing studies generally focused on the experimental classifiers developed for research purposes only. The security of widely-used classifiers in commercial products still remains unclear. From a practical point of view, evaluating the security of commercial classifiers is more significant for protecting end users from evasion attacks.

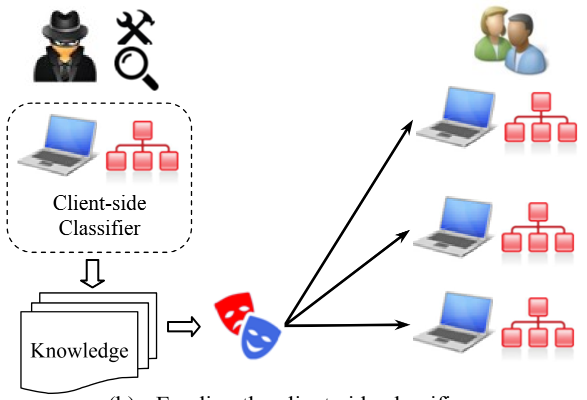
To this end, in this study, we investigate the security challenges for the client-side classifiers via a case study on the *Google's phishing pages filter* (GPPF), a very widely-used classifier for automatically detecting unknown phishing pages. The classifier is integrated within the *Chrome* browser and is invoked for every web page visited by users to check whether it is phishing. Due to the popularity of *Chrome*, there are over one billion users using GPPF against potential phishing attacks [2]. It is also probably the most widely-used classifier as we know. If the adversary can easily evade it, countless users will be exposed to out-of-control phishing attacks.

In this paper, we demonstrate a practical and effective attack methodology, named *classifiers cracking*, in which various reverse engineering techniques are leveraged to extract sufficient knowledge from the client-side classifier for launching evasion attacks. Specifically, via some static and dynamic analysis on the implementation of *Chromium* (the development version of *Chrome*), we successfully extract the classification model of GPPF from *Chromium*. The extracted model mainly involves the classification algorithm, the 2,130 scoring rules and their corresponding weights, as well as the 1,009 hashed features composing the scoring rules. With the help of some public datasets (e.g., large corpora), we then launch a collision attack to the hashed features and decrypt 815 (80.8%) of them only within a dozen of hours. As a result, we can completely reverse engineer 1807 (84.8%) scoring rules, covering most of the high-weighted rules. Additionally, 196 (9.2%) scoring rules are partially cracked and can also be exploited to compromise the classification. There are only 127 (6.0%) rules surviving from the collision attack.

Based on the cracked information, we design two kinds of evasion attacks, i.e., *good features insertion* and *bad features elimination*. The basic idea behind them is to add or to remove some features with remarkable contributions to GPPF scoring into or from the target phishing pages to reduce their phishing scores, making the computed scores lower than the positive threshold defined by



(a) Evading the server-side classifier



(b) Evading the client-side classifier

Figure 1. Threats to classifiers.

GPPF. We evaluate the effectiveness of the attacks on the 100 latest real phishing pages collected from PhishTank [3], a famous phishing URLs tracking site. The results show that we can easily manipulate all the phishing pages under the direction of the cracked knowledge, to make them successfully evade the detection of GPPF in the latest version of *Chrome*.

We also analyze the effectiveness of existing defense techniques when applying them to client-side classifiers. To the best of our knowledge, there is still lack of a perfect approach to protect client-side classifiers from being cracked. We believe that how to protect the client-side classifiers is still an open problem.

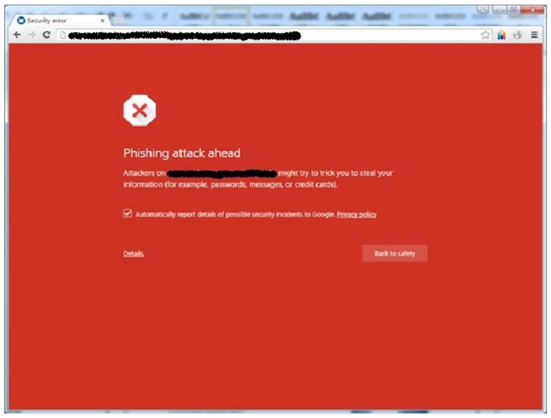
This paper makes the following two main contributions.

- We propose a new attack methodology, *classifiers cracking*, aiming at the client-side classifiers. The adversary can follow it to readily acquire exploitable knowledge from the target classifier to launch effective evasion attacks.
- We successfully crack and evade the GPPF, a commercial classifier with over one billion users. It demonstrates that the existing client-side classifiers are indeed vulnerable to the classifiers cracking attacks.

2. BACKGROUND

2.1 Threat Model

As shown in Figure 1(a), how to classify an instance in a server-side classifier is often a black-box to the adversary. The adversary can only send some queries and analyze responses to learn the information about it. In many cases, this is already enough to launch an evasion attack. The adversary can construct a malformed instance to fool the classifier based on the information learned in advance.


Security error

[redacted]

Phishing attack ahead

Attackers on [redacted] might try to trick you to steal your information (for example, passwords, messages, or credit cards).

[✓] Automatically report details of possible security incidents to Google. Privacy policy

Details

Back to safety

Figure 2. Phishing warning page.

However, when a classifier is deployed in the client sides, the situation may become worse. As shown in Figure 1(b), for a client-side classifier, its operations are performed in a white-box. The adversary can leverage almost all kinds of analysis techniques, such as debugging, disassembling, code analysis, dynamic taint tracking, to thoroughly analyze the target classifier. As a result, the adversary has an opportunity to get more comprehensive knowledge about the classifier so as to develop more sophisticated evasion attacks. The malformed instance can be applicable for all the users using the classifier. Besides, if the adversary gets perfect knowledge about the classifier, she can even reengineer a new classifier for commercial purposes. In this study, we assume that all the implementation and configuration of the client-side classifier are available for the adversary. The adversary can figure out the type of classification model, the classification algorithm and the feature extraction method by leveraging various techniques. Considering the advancement of modern analysis techniques, this assumption is reasonable.

Some client-side classifiers have already introduced some defense techniques to prevent the adversary from learning crucial information. For example, GPPF employs the cryptography technique to protect the classification model. Unfortunately, it is proved to be ineffective to against classifiers cracking (discussed in Section 3 and 4).

2.2 Phishing and GPPF

According to the latest report [4] of Anti-Phishing Working Group (APWG), phishing attacks remain widespread: the number of unique phishing reports submitted to APWG during Q4 of 2014 was 197,252, and increased by 18% on top of the 163,333 received in Q3. To minimize the impact of phishing attacks, a variety of methods have been proposed to detect phishing pages, involving machine learning [37][53][58] or other techniques [26][27][29][31][33][46][60][61].

Modern web browsers also provide detection tools to assist end users against phishing attacks. In *Chrome*, Google provides not only the blacklists of malicious URLs but also a trained classifier (GPPF) which automatically detects phishing pages as a countermeasure to the phishing problem [5]. These mechanisms serve as a guard when a request comes, and the request URL will be checked before the content is allowed to begin loading. The URL is checked against two blacklists: malware and phishing. If the URL is matched with any one of the two blacklists, *Chrome* will block the request and jump to a warning page as shown in Figure 2. More importantly, for the URL not matched in the blacklists, *Chrome* will further invoke GPPF to determine whether the URL is legitimate or phishing. In practice, the phishing blacklist needs to be updated constantly. The browser may be vulnerable to newly created phishing websites. GPPF acts as an indispensable role in protecting end users from unknown phishing pages.

In practice, GPPF is trained offline. Google collects massive pages from various domains as the training dataset. The adversaries have no opportunity to alter the training dataset enough to fool the trained classifier to misclassify phishing pages as legitimate ones. However, as an internal component of the *Chrome* browser, GPPF is completely deployed and running in the user environment. This actually allows the adversary to freely analyze its implementation and configurations to construct more sophisticated phishing attacks.

According to the report of StatCounter [6], from Aug 2014 to Aug 2015, *Chrome* shared an average of 48.6% market and was the most popular web browser. In May 2015, Google announced that *Chrome* has over one billion active users [2]. This means over one billion users' web surfing are protected by GPPF. Hence, we have reason to believe that the security breach of GPPF can result in a significant risk that should not be neglected.

3. CRACKING GPPF

There is very limited public information about the design and implementation of GPPF. We choose to directly analyze the development version of the *Chrome* browser, *Chromium*, to crack GPPF. The cracking includes two main steps: (1) extracting the classification model of GPPF from *Chromium*; and (2) decrypting the hashed features of the model. **It needs to be mentioned that some sensitive details of the cracking are intentionally omitted to prevent them from being used for malicious purposes.**

3.1 Extracting the Classification Model

3.1.1 Classification Algorithm

The multi-process architecture that *Chrome*/*Chromium* adopts helps it to be more robust. According to a very brief description [5], we can know that *Browser* process will periodically fetch an updated model from Google's server and send it to every *Render* process via an IPC channel. This allows the classification to be done in the Render process, which will score the request page to tell whether it is phishing or not.

The reverse engineering technique is employed to extract encrypted classification model features from *Chromium*. Although *Chromium* is an open-source project, it is difficult to directly extract the encrypted model only by statically analyzing its source code. Instead, we performed a hybrid static and dynamic analysis on the *Chromium* to find the scoring point and extract the encrypted model.

By statically analyzing the source code of *Chromium* and comparing the rendering processes of a phishing page and a legitimate page with a debug tool *gdb*, we locate and confirm the GPPF's scoring function *ComputeScore*(), which is a method of the *Scorer* class defined in the file *scorer.cc*. Combining a dynamically backward tracking of the execution path started from *ComputeScore*() and a static analysis on the source code, we conclude the workflow of the classification. As shown in Figure 3, first, the classifier extracts three kinds of page features from the current web page in the order of *URL*, *DOM* and *Term* features. Second, the collected page features are hashed with the SHA-256 algorithm and are sent to the function *ComputeRuleScore*() to

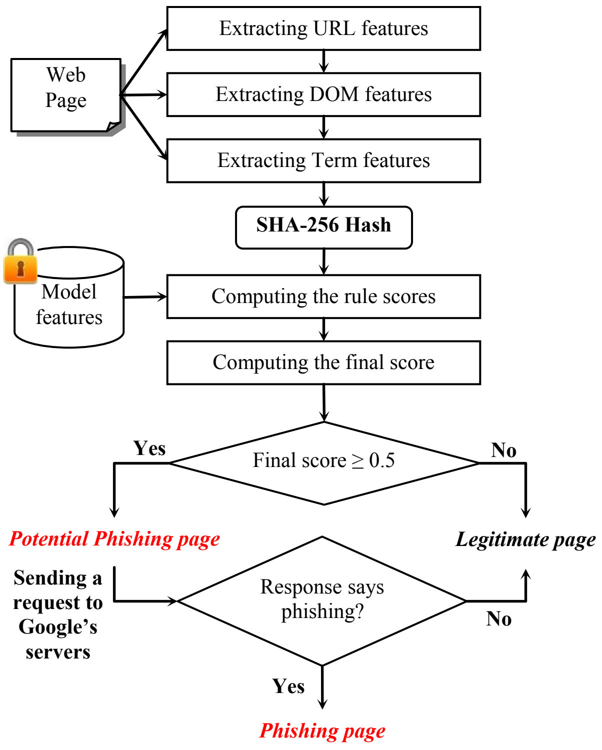


Figure 3. Classification workflow.

compute the rule score for every scoring rule, along with the hashed model features. Third, *ComputeScore*() combines all the rule scores to generate a final score for the current page. Finally, the score is compared with a predefined threshold (fixed as 0.5). If the score is smaller than the threshold, the page will be identified as legitimate; otherwise, it will be regarded as a potential phishing page. The browser will send its URL to the Safe Browsing servers to check the URL against a remote blacklist. The servers may comprehensively analyze the page, e.g., feeding it to the internal phishing page classifier. If the response from the Safe Browsing servers identifies the URL as phishing, the page will be blocked. Consequently, if we can decrease the score of a phishing page, it can escape from being checked and analyzed by Google's servers, and be directly regarded as a legitimate one.

Based on the analysis of the scoring process, we find the GPPF is a *logistic regression* classifier, which uses the following two expressions to compute the phishing score for the target page. GPPF computes the total score for the page in log odds using the expression (1), and uses the normalization expression (2) to transform the score in log odds to the final score.

$$Logodds = W_1 + \sum_{i=2}^{2130}\left(W_i \prod_{j=1}^{n_i} V_{i,j} \right) \qquad (1)$$

$$score = \frac{e^{Logodds}}{1+e^{Logodds}} \qquad (2)$$

According to expression (1), the computing of the log odds of the page involves 2,130 scoring rules. Every rule has a weight, namely $W_1 \sim W_{2130}$. Except for the first rule, every rule consists of one to four (i.e., n_i for the ith rule) model features. Before computing the rule scores, the page features are first mapped into string forms, which will be hashed and compared with the model features. For every rule, the classifier creates a set of feature values (i.e., $V_{i,1} \sim V_{i,n_i}$) for all matched model features. For

Table 1. Scoring rule examples

Rule	Features #	Hashed Feature	Weight
R_{1494}	2	32ffbec120ed857f57f3d7bb37 e6652955b21da7a7efd81d9a9 aa2865173eb35	-1.26907706
		ec92914c7db4483437c84975 8c45cf8bbc6dd0148cdb2f72b ec0a728e8c91a7d	
R_{2050}	1	760e98536a709d0fcb9b717eb 542cc5af77bbabf60a501dbf7d f81a111d1e807	2.5238471

Boolean feature, *True* is converted to 1.0 and *False* is converted to 0.0. The continuous features are scaled to be between 0.0 and 1.0. If a model features of the rule is absent from the target page, its feature value will be set to 0.0. The score of the rule will be computed by combining the product of all the feature values and its weight. Finally, the log odds of the page will be produced by summing up all the rules scores.

To crack the classification model, we need to recover the weight and model features for every scoring rule. The rule weight can be collected by debugging the Render process. We set a breakpoint in *ComputeRuleScore*(), in which an extractor written in *gdb* script is invoked to read the weight information from the rule objects in the memory and save them in a file. In a similar way, we also get the number of model features for every rule. However, the model features are not stored in plaintext; instead, they are hashed with the SHA-256 algorithm and are hidden in some complex data structures. With carefully tracking of the scoring process, we locate their addresses and design a *gdb* script to extract them from rule objects. Take two extracted rules as examples. As listed in Table 1, the rule R_{1494} is a negative rule with two features. This kind of rule is used to identify the good property indicative of legitimate pages. On the contrary, the rule R_{2050} is a positive rule, including only one feature. Some of the model features are presented in different scoring rules. After eliminating duplicates, in total, we collect 1,009 individual hashed model features. The decryption of them will be described in Section 3.2.

3.1.2 Model Features

To decrypt the hashed model features, we should first get clear about their semantics and how the page features are mapped into them. When computing the score, three kinds of page features will be mapped into corresponding model features in different ways.

URL features. In practice, the phishers often obfuscate their URLs to hide suspicious addresses or confuse victims into believing they come from a trusted party. Based on the observation, in GPPF, some characteristics of the URL are employed to identify phishing pages. By analyzing the implementation of the classifier, we recover all seven kinds of properties of the URL being extracted as the page URL features, as shown in Table 2. The page URL features will be converted into string forms, which will be hashed and compared with the encrypted model URL features during computing scores.

The page URL features can be categorized into two groups. For one of the first group of features (the first three in Table 2), if it is present in the URL, a hashed predefined string will be taken as its corresponding model feature. Take the first page URL feature as an example. If the hostname part of the URL is a numeric IP address, the string "*UrlHostIsIpAddress*" is hashed with the SHA-256 algorithm to act as the model feature. For the second group of

Table 2. URL features

No.	Page URL Features	Model URL Features
1	The hostname is an IP address?	*UrlHostIsIpAddress*
2	The number of other host components is greater than one?	*UrlNumOtherHostTokens>1*
3	The number of other host components is greater than three?	*UrlNumOtherHostTokens>3*
4	Top level domain	*UrlTld=**
5	The first host component below top level domain	*UrlDomain=**
6	Other host components	*UrlOtherHostToken=**
7	Path token in URL	*UrlPathToken=**

features (the last four in Table 2), a string in equation form will be generated by concatenating a predefined string and the concrete URL property. For example, for the fifth page URL feature, if the URL is *www.phishing.com*, the string "*UrlDomain=phishing*" will be hashed as the model feature. In scoring rules, all the URL features will be assigned a Boolean feature value, i.e., 1.0 if it is present in the page or 0.0 if it is absent.

The predefined strings used to generate the model feature (shown in the third column of Table 2) can be inferred from the implementation of the classifier. However, we cannot directly recover the complete plaintexts from the hashed model URL features in equation forms. In GPPF, there are hundreds of model features about the URL in equation forms. Based on their semantics discussed above, we design a collision attack to decrypt them as far as possible (described in Section 3.2).

DOM features. GPPF also uses some features about the Document Object Model (DOM) elements of the page to tell whether it is phishing. As shown in Table 3, we recover all 12 kinds of DOM features employed by GPPF. In a similar way to the URL features, these page DOM feature will also be converted to string forms.

As listed in Table 3, the first seven page DOM features are used to identify the structure property of the page, e.g., to determine whether the page has some kinds of DOM elements or not. These features directly correspond to seven predefined strings respectively, which will be hashed and compared with the model features. For example, if the page has the *<form>* element, the string "*PageHasForms*" will be hashed to act as the corresponding model feature. The eighth page DOM feature records all external domains that the page links to, which will be mapped into a string in equation mode for every individual external domain. In scoring rules, all the above DOM features will be assigned a Boolean feature value. The last four page DOM features indicate the fraction of some certain kinds of DOM elements. They correspond to four predefined strings. In scoring rules, the values of matched features are set to the fraction value scaling between 0.0 and 1.0.

For the DOM features, the related predefined strings can be directly recovered and are shown in the third column of Table 3. For the eighth page DOM feature, there are many related hashed model features in equation forms to identify different external domains. A collision attack is performed to recover their plaintexts (described in Section 3.2).

Table 3. DOM features

No.	Page DOM Features	Model DOM Features
1	Page has <form> element?	*PageHasForms*
2	Page has <input type=text> element?	*PageHasTextInputs*
3	Page has <input type=password> element?	*PageHasPswdInputs*
4	Page has <input type=radio> element?	*PageHasRadioInputs*
5	Page has <input type=checkbox> element?	*PageHasCheckInputs*
6	The number of <script> elements in the page is greater than 1?	*PageNumScriptTags>1*
7	The number of <script> elements in the page is greater than 6?	*PageNumScriptTags>6*
8	Token feature containing each external domain that is linked to	*PageLinkDomain=**
9	Fraction of form elements whose action points to an external domain	*PageActionOtherDomainFreq*
10	Fraction of links in the page which point to an external domain	*PageExternalLinksFreq*
11	Fraction of page links that use https	*PageSecureLinksFreq*
12	Fraction of images whose src points to an external domain	*PageImgOtherDomainFreq*

Term features. In GPPF, the terms appearing in the page are taken as a kind of feature. A term feature can be a single word or a compound of multiple words (at most five).

When fetching the page terms, the page text is first converted into a list of words in lowercase. In practice, using every word of the page text to construct features will greatly overburden the learning process. Instead, GPPF only handles the words contained in a predefined set. A fast hash algorithm, *Murmurhash3*, is employed to implement a word filter. GPPF maintains a list of candidate words, which are hashed with the Murmurhash3 algorithm. It was generated by collecting the words with the highest term frequency-inverse document frequency (TF-IDF) values [50] from a large dataset.

GPPF uses an array named *previous_words* to construct the page term features. The array can store at most five continuous candidate words of the page text and is initially empty. The first word is fetched and removed from the page word list. Its Murmurhash3 value is computed to determine whether it is contained in the candidate list or not. If it is a candidate, the word will be added in the first element of *previous_words*. GPPF then checks the subsequent word in the list and adds it to the array in sequence if it is also a candidate word. Whenever a word is added, all words currently contained in the array (at most five) are connected and combined with a predefined prefix ("*PageTerm=*") to construct a phrase. It will be hashed with SHA-256 algorithm and compared with the hashed model term features. For example, if three continuous words ("*abc*", "*def*", and "*ghi*") have been added in the array, the generated corresponding phrases will be "*PageTerm=abc*", "*PageTerm=abc def*", and "*PageTerm=abc def ghi*". In scoring rules, the values of a term feature will be set to 1.0 if there is a matched phrase; otherwise to 0.0. When encountering a non-candidate word or the array is full, GPPF will

Table 4. Model features needed to be decrypted

Category	Model Features	Total	Decrypted		%
URL-related	*UrlTld=**	563	69	426	75.7%
	*UrlDomain=**		21		
	*UrlOtherHostToken=**		28		
	*UrlPathToken=**		201		
	*PageLinkDomain=**		107		
term-related	*PageTerm=**	432	375		86.8%
Sum		**995**	**801**		**80.5%**

clear the array, fetch the next word and repeat the above steps until the list is empty.

In GPPF, there are 432 hashed model term features. Every one corresponds to a phrase that may consist of one to five words. We also use a collision attack to recover their plaintexts.

3.2 Collision Attacks

As discussed in Section 3.1, besides 14 features being directly recovered in the model extraction, there are still 995 hashed model features needed to be decrypted. As shown in Table 4, they can be divided into two categories: *URL-related* and *term-related*. According to their semantics, we design different collision attacks to decrypt them.

3.2.1 Decrypting URL-related Features

In total, there are 563 hashed URL-related features. So far, it is impossible to directly construct a collision for a given SHA-256 hash value. Instead, we collect four datasets related to URLs to perform targeted brute force attacks to find potential collisions as much as possible. To prevent the adversary from reproducing the attacks, the sources of the datasets are not presented in this paper.

1) We use a dataset with about 8,000 top level domain names to decrypt *UrlTld* features. We select the name from the set one by one and add the prefix "*UrlTld=*" to generate a test case. By hashing it with SHA-256 and comparing the hash value with all URL-related features, we successfully recover 69 *UrlTld* features with a desktop computer in about five minutes.

2) We collect over 30,000 URLs of history phishing pages, and use the different elements of the URLs (e.g., hostname) to generate test cases for other four kinds of URL-related features. In a similar way as above, 171 features are successfully decrypted in about four minutes, including 20 *UrlDomain*, 27 *UrlOtherHostToken*, 17 *UrlPathToken* and 107 *PageLinkDomain* features.

3) With the URLs of legitimate pages in thousands of top sites, we get 3 *UrlDomain* features and 34 *PageLinkDomain* features in less than one minute.

4) A very large URL database with over 2,000,000 records is leveraged to construct test cases. The decryption process takes about 20 minutes. As a result, we get 46 *UrlTld*, 21 *UrlDomain*, 28 *UrlOtherHostToken*, 201 *UrlPathToken* and 107 *PageLinkDomain* features.

After removing duplicates, as listed in Table 4, we eventually recover a total of 426 (75.7%) URL-related features, including 69 *UrlTld*, 21 *UrlDomain*, 28 *UrlOtherHostToken*, 201 *UrlPathToken* and 107 *PageLinkDomain* features.

Table 5. Decrypting the term features with seven corpora

Language	Decrypted	Time
English	201	1.7 hours
French	6	2.3 hours
German	51	3.2 hours
Spanish	5	1.1 hours
Dutch	1	6 minutes
Chinese	27	20 minutes
Japanese	1	5 minutes
Sum	**292**	**8.8 hours**

3.2.2 Decrypting Term-related Features

GPPF employs 432 hashed term features to detect phishing pages based on the page text. In practice, the text of a phishing page can be written in various languages. To this end, we collect some full-text corpora for seven popular natural languages (English, French, German, Spanish, Dutch, Chinese and Japanese) to perform collision attacks. The basic steps are as follows.

- According to the semantics of the term feature, we build a candidate word filter based on the implementation of the Murmurhash3 algorithm in *Chromium*. With it, we extract all possible word sequences consisting of one to five continuous candidate words from these corpora respectively.
- For every word sequence, adding the prefix "*PageTerm=*" to generate a test case.
- Hashing every test case with SHA-256 and comparing the hash value with all term features to find potential collisions.

Via the above steps, we successfully recover 292 (67.9%) term features in various languages in about 8.8 hours. The result is detailed in Table 5.

To further improve the cracking result about term features, we also perform blind brute force attacks. We construct an alphabet consisting of letters in western languages. With the alphabet, all possible combinations of no more than eight letters are produced. After filtering, they are used as candidate words to generate test cases to find collisions. Surprisingly, in about 16 hours, we recover 281 term features only using a part of test cases. In a similar way, we also quickly recover 40 term features based on a set of Chinese, Japanese and Korean (CJK) ideographs. The related results are detailed in Table 6 and Table 7 respectively.

After combining all above attacks results and removing duplicates, we eventually recover a total of 375 (86.8%) term features.

Table 6. Decrypting the term features with an alphabet

Term Size	Candidate Words	Decrypted	Time
1-word	1-letter to 8-letter	186	1 minute
2-word	1-letter to 8-letter	76	8.6 hours
3-word	1-letter to 6-letter	15	14 minutes
4-word	1-letter to 4-letter	4	7.4 hours
Sum		**281**	**16.25 hours**

Table 7. **Decrypting the term features with CJK ideographs**

Term Size	Candidate Words	Decrypted	Time
1-word	1-ideograph to 3-ideograph	31	1 minute
2-word	1-ideograph to 3-ideograph	7	< 1 minute
3-word	1-ideograph to 3-ideograph	2	< 1 minute
4-word	1-ideograph to 3-ideograph	0	2 minute
Sum		**40**	**5 minutes**

3.3 Result Analysis

As shown in Table 4, we successfully decrypt 801 (80.5%) model features with collision attacks. Together with 14 features being recovered in the model extraction, we eventually get the complete plaintexts of a total of 815 (80.8%) model features.

After applying the decryption result to 2,130 extracted scoring rules, we can completely reverse engineer 1807 (84.8%) rules (every feature of them is decrypted). Besides, there are also 196 (9.2%) rules we cannot completely crack, but at least one of their features is decrypted. Only 127 (6.0%) rules remain confidential, no one of their features is cracked.

According to their weights, GPPF's scoring rules can be categorized into two types: *positive rules* and *negative rules*. As their names suggest, the former are assigned with a positive weight and can cause a rise in the phishing score for the page, while the latter are just the opposite. Naturally, the top-weighted positive or negative rules will make remarkable contributions to tell whether a page is phishing. After analyzing top 100 most weighted positive rules, we learn that 66 of them are completely reverse engineered, and 20 are partially cracked. For the top 100 most weighted negative rules, 77 of them are completely reverse engineered, and 21 are partially cracked. In other words, given the cracking result, the adversary has a great chance to disguise a phishing page as a legitimate one by targetedly manipulating its content.

4. EVASION ATTACKS

In this section, we perform some evasion experiments to demonstrate the effectiveness of the classifiers cracking via exploiting the recovered knowledge.

For a specific phishing page, we can infer which features can be added or removed to reduce its phishing score based on the cracking result presented in Section 3. If a feature can provide negative contributions to the phishing scoring for a page, we call it as a *good* feature from the adversary's point of view. On the contrary, if a feature only has positive contributions, we call it a *bad* feature. Correspondingly, we design two kinds of evasion attacks, *good features insertion* and *bad features elimination*. The basic idea behind them is to add or remove appropriate good or bad features into or from a phishing page to make its phishing score lower than the threshold, resulting in a misclassification. The latest 100 real phishing pages are collected from PhishTank as the attack dataset. We will try to use the two evasion attacks to manipulate them to evade the detection of GPPF. To minimize the potential side-effects, we will use pseudonyms when referring to specific good features or bad features in the following part of this section.

Table 8. The required number of Good features

Feature	MIN	MAX	Average
URL	1	10	2.5
DOM	1	6	2.2
Term	1	17	3.7

4.1 Good Features Insertion

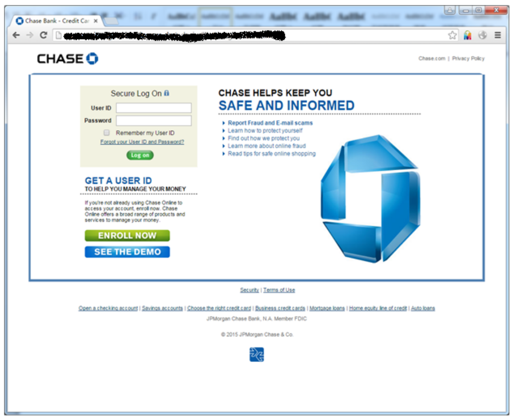
Given a phishing page, there may be many features that can be leveraged to reduce its phishing score. By utilizing plenty of negative rules having been completely reverse engineered, we can adopt a very primitive but effective way to choose desirable good features. In fact, we can sort all negative rules only with one recovered feature by their weights, and directly use the features of top-weighted rules as good feature candidates for all target pages. More surprisingly, for many phishing pages in the dataset, we can easily convert them to legitimate pages only by inserting just one such good feature. Moreover, as detailed in Table 8, we find that it is also effective to use only one kind of good feature. For example, we can reduce the scores of all test pages to be lower than 0.5 by inserting at most six good DOM features into the page. On average, 2.2 good DOM features are required.

It should be noted that a sophisticated adversary can carefully introduce the good features to preserve the utility of phishing pages. For example, to prevent the inserted terms from attracting the attention, their color can be set as the background color.

After introducing above good features, the manipulated test pages are deployed in our Web server. We then use the latest version of *Chrome* (45.0.2454) to visit them one by one to check whether they can successfully evade the detection of GPPF. We find all the dressed-up pages (100%) are regarded as legitimate pages and display properly in the browser. For example, there is a phishing page which imitates the login page of Chase Bank. When browsing it, *Chrome* can successfully block it as a phishing page and jump to the warning page as shown in Figure 2. In fact, the page is given a very high phishing score 0.9986. However, after inserting six good term features *T*1 ~ *T*6 into its text, the score is reduced to only 0.2784. As a result, the dressed-up page can be normally visited with *Chrome* as shown in Figure 4.

4.2 Bad Features Elimination

Compared with the good features insertion, selecting proper bad features from a given phishing page to perform an effective evasion attack is not a trivial task. The number of available bad


Chase Bank - Credit Car
[redacted]
CHASE
Chase.com | Privacy Policy
Secure Log On
User ID
Password
Remember my User ID
Forgot your User ID and Password?
Log on
CHASE HELPS KEEP YOU
SAFE AND INFORMED
Report Fraud and E-mail scams
Learn how to protect yourself
Find out how we protect you
Learn more about online fraud
Read tips for safe online shopping
GET A USER ID
TO HELP YOU MANAGE YOUR MONEY
ENROLL NOW
SEE THE DEMO
Security | Terms of Use
Open a checking account | Savings accounts | Choose the right credit card | Business credit cards | Mortgage loans | Home equity line of credit | Auto loans
JPMorgan Chase Bank, N.A. Member FDIC
© 2015 JPMorgan Chase & Co.

If you're not already using Chase Online to access your account, enroll now. Chase Online offers a broad range of products and services to manage your money.

Figure 4. The dressed-up phishing page can evade GPPF.

features is limited for a given page. Additionally, some features can be referred by multiple scoring rules. A feature may be not only present in a positive rule but also in a negative rule. Directly removing the features in positive rules may also result in some negative rules losing their efficacy.

To this end, we design a search-based method to automatically select proper bad features for a given page. Specifically, we implement a script to compute the contribution of one feature or a set of features to the final score, by removing the feature or the set from the page and re-computing the score. For a given page, we apply the script to all its recovered features to search for a feature or a set of features whose contribution is enough for the exploitation. In other words, after removing the feature or the set of features, the score of the page will become lower than the threshold, allowing it to be classified as a legitimate page.

With the method, we successfully find proper bad features for every test page respectively. By eliminating corresponding bad features from the pages respectively, all test pages (100%) can evade the classification and be normally rendered by *Chrome*. Take the phishing page shown in Figure 4 as an example. We find four bad term features $BT1 \sim BT4$ for the page and eliminate them with some obfuscation techniques, such as changing a word from singular to plural form. As a result, we successfully reduce its score from 0.9986 to 0.4591 and dress it up as a legitimate page. In the experiment, we find that removing at most five bad features is enough to make the page evade the classifier. On average, 3.1 bad DOM features are required.

5. MITIGATION

Google developers have discussed the potential adversarial attacks that their server-side phishing page classifier might encounter [58]. They believe that possible attacks on the classifier are either limited or expensive. From their points of view, the adversary who tries to evade the classifier by disguising the phishing page as a legitimate one cannot preserve both its utility and visual similarity at the same time. However, the assumption is incorrect to the client-side classifier. Thanks to the cracking results, we can purposely introduce some easy-to-hide good features to evade GPPF with a very low cost. For example, we are able to make the newly added term features invisible by setting their color to be the same as the background color of the target page.

In practical applications, the phishing page classifier is proved to be a very valuable tool against phishing attacks under the non-adversarial environment. Tens of thousands phishing sites are detected by Safe Browsing per week [7]. To this end, the developers may want to improve its robust as well as change the architecture as little as possible. A natural and direct idea is to select the features difficult to be recovered by brute force attacks. For example, the developers can just select the comparative long phrases, 5-word phrases or even longer, as the term features. This would result in a combinatorial explosion when the adversary performs a blind brute force attack for cracking. The computation of enumerating and hashing all possible compounds of five words is unacceptable. Unfortunately, this idea is not effective enough if the adversary is aware of the feature extraction method. In fact, the adversary can still reverse engineer sufficient features by collecting appropriate page-related data as test cases to perform collision attacks. The adversary can take the data as a web page to extract the possible word sequences according to the feature extraction method and then hash the sequences to check whether they are term features. For a concrete feature extraction method, the amount of sequences is actually limited regardless of how long the sequence is. Given appropriate test cases, the adversaries have a fair chance to find sufficient collision instances. As presented in Section 3.2.2, we recover 292 (67.9%) term features only using seven full-text corpora in 8.8 hours. These features are already enough for evasion attacks.

Based on the above discussions, we can learn that the most effective form of defense is to essentially increase the complexity of reverse engineering the classification model, especially the semantics of features. A direct approach is to convert the classifier to a closed-source component of the system. This will improve the security of the classifier to some extent. The adversary cannot easily understand the implementation of the classifier by reading its source code. However, it is not enough when the adversary leverages some modern analysis techniques, e.g., [11][24][51][56], to extract exploitable knowledge from binary-format software. A more sophisticated approach is to employ the code obfuscation technique to conceal the logic of the classifier as far as possible. Although the complexity and the cost of analyzing code are remarkably improved, cracking still cannot be thoroughly prevented by using code obfuscation. In fact, some deobfuscation techniques have been developed to reverse engineering obfuscated code [55][59]. Because the client-side classifier does run in the environment may be completely controlled by adversaries, they always can find a way to observe its running and gather exploitable information.

In conclusion, we believe that how to protect the client-side classifiers still remains an open problem.

6. DISSCUSSION

In this study, we present an attack methodology, classifiers cracking, aiming at client-side classifiers and successfully demonstrate its effectiveness with a widely-used classifier, GPPF. In theory, the methodology is generic and applicable to other client-side classifiers. However, when applying the methodology to a specific classifier, we need to develop a specially designed crack techniques according to its implementation. In fact, there are many classifiers equipped with different classification algorithms, e.g., [8][38]. To further demonstrate the security challenges brought by classifiers cracking, in the future, we will pay attention to some other types of classifiers and investigate their security from the point of cracking. These classifiers may take security into consideration to different extents and be deployed in different ways. More reverse engineering techniques may need to be employed to crack them.

As described in Section 3 and 4, we eventually completely reverse engineer 84.8% scoring rules of the GPPF classification model, which is proved to be sufficient for launching effective evasion attacks. However, in fact, we can get better cracking results by introducing more appropriate corpora. For example, using a comprehensive database of history phishing pages can decrypt more term features. Sometimes, the adversary may want to get perfect knowledge about a classifier for some special purposes, such as stealing its techniques to reengineer a new classifier. Besides, it needs to be emphasized that some seemingly unrelated dataset, e.g., a corpus, can also be leveraged to compromise the security of client-side classifiers. The developers should collect as much as possible datasets, especially publicly available, to evaluate the robust of their classifier before releasing it.

We have got sufficient knowledge about the GPPF classification model by cracking it. This allows us to easily find exploitable good and bad features for a given page. In this study, it is not necessary to design a sophisticated algorithm to more effectively and efficiently find exploitable features. However, if the adversary has only limited knowledge about the target classifier, she can develop a powerful algorithm to discover exploitable features. Furthermore, in theory, combining the good features insertion and bad features elimination can produce better performance. It is also helpful for the adversary to attack a classifier. To this end, developers should prevent the information of their classifier from being inferred by the adversary as far as possible.

7. RELATED WORK

Many existing studies have paid much attention to the security of classifiers, and the arm race between adversaries and defenders will never end.

Attacks on Classifiers. The attacks can be categorized into two types by their influences: causative attacks and exploratory attacks.

In causative attacks, the adversary has the chances to affect the training process by contaminating training data (e.g., injecting many specially crafted samples). This kind of attack has been used to degrade the performance of a lot of learning-based applications, such as biometric authentication [17][19], spam filtering [44], and network intrusion detection [34][49]. In [17], a method is proposed to mislead an adaptive biometric system to perform self-update by submitting a proper sequence of spoofed biometric traits to the sensor and cause a misclassification eventually. A further work [19] reveals that poisoning attacks can be used to compromise face templates in a more general case. Another study [44] succeeds in exploiting machine learning to compromise a spam filter by manipulating the filter's training data. They proposed two kinds of poisoning attacks by inserting different sets of words into attack emails: *dictionary attacks* inject words indicative of legitimate emails to increase misclassifications, and *focused attack* tries to introduce words to have the filter block one specific kind of emails (e.g., emails from business rivals). Besides, the intrusion detection systems may also be vulnerable to causative attacks [21][34][49]. The adversary can inject carefully crafted malicious traffic samples into training dataset and finally force the classifier to learn a wrong model of the normal traffic.

In exploratory attacks, the adversary tries to figure out as much knowledge (e.g., type of classifier, features, and threshold) of the classifiers as possible to effectively evade them. Exploratory attacks have been applied to various security applications. Lowd and Meek conduct an attack that minimizes a cost function [40]. They further propose attacks against statistical spam filters that add the words indicative of non-spam emails to spam emails [39]. The same strategy is employed in [44]. In [41], a simple but effective attack methodology called *reverse mimicry* is designed to evade structural PDF malware detection systems. The main idea is injecting malicious content into a legitimate PDF while introducing minimum differences within its structure. The related experiments show that some very popular classification algorithms (e.g., SVMs and neural networks) can also be evaded with this method. A recent work [52] uses PDF_{RATE}, an online learning-based system for detection of PDF malware, as a case to investigate the effectiveness of evasion attacks. The study reconstructs a similar classifier through training one of the publicly available datasets by a few deduced features, and then evades PDF_{RATE} by insertion of dummy content into PDF files. Additionally, in [18], a simple algorithm is proposed for evasion of classifiers with differentiable discriminant functions. The study empirically demonstrated that very popular classification algorithms, e.g., SVMs and neural networks, can still be evaded with high probability even if the adversary can only learn limited knowledge.

Unfortunately, to our best knowledge, all of the existing studies don't pay any special attention to the client-side classifiers. As demonstrated in this study, the client-side classifiers have a larger attack surface and hence suffer from a larger number of potential attacks. The main contribution of this paper is revealing a very serious security threat to client-side classifiers. By employing some traditional reverse engineering and cracking techniques, such as the dynamic debugging and collision attacking, the adversary can extract enough knowledge from the implementation of the classifier to launch effective evasion attacks.

Hash Cracking. Hash cracking technique is mainly used in inverting hashed passwords, including brute force attacks, dictionary attacks, and rainbow table attacks. In the brute force attack [1][35], the cracker will compute the hashes for all possible password candidates and compare them with the given hashed password. If there is a matched one, the plaintext of the password is found. Although it is easy to implement, the brute force attack is time-consuming. The dictionary attack [42][43][57] is a more intelligent variant of a straight brute force approach. It utilizes a dictionary of words to compute hashes and compares them with the given hash. This attack is usually fairly efficient and requires much less time than a brute force. However, if the password is not present in the dictionary, the attack will fail. The rainbow table attack [45] is a more sophisticated hash cracking method. Cracker will pre-compute all plaintext/hash pairs and store them in a file called rainbow table. The computation is also time-consuming. However, given the rainbow table, the cracker can immediately find the plaintext of a hashed password via a table query.

In this paper, we take some corpora as the cracking dictionaries to recover as many hashed model features as possible. Besides, to improve the recovery rate, we further launch a brute-force attack to find model terms by generating possible terms with a western alphabet and a set of CJK ideographs.

Dynamic analysis. Some existing approaches adopt dynamic analysis techniques to reverse-engineer the implementation of a software. Polyglot [24] uses dynamic binary analysis to extract the protocol message format used by a target software. Howard [51] instruments the QEMU [11] processor emulator to extract critical data structures from a software. TaintScope [56] performs a differential analysis on the branch instruction traces of program executions with well-formed and malformed inputs, to identify the checksum check point.

Since *Chromium* is an open-source project, we can compile it to get a debug version with symbols of functions, variables and data structures. As such, we can directly use a debug tool (e.g., *gdb*) to locate the scoring point and identify critical data structures of the phishing model. If the target system to be cracked is not open-source, crackers can employ the above dynamic analysis techniques to achieve the same goal.

Defenses. Many countermeasures against evasion attacks have been proposed, such as using game theory [22][23] or probabilistic models [16][48] to predict attack strategy to construct more robust classifiers, employing multiple classifier systems (MCSs) [13][14][15] to increase the difficulty of evasion, and optimizing feature selection [30][36] to make the features evenly distributed.

Game-theoretical approaches [22][23] model the interactions between the adversary and the classifier as a game. The adversary's goal is to evade detection by minimally manipulating the attack instances, while the classifier is retrained to correctly classify them. However, the retraining procedure is very expensive in the situation where the classifier is cracked. The adversary always can construct an attack instance to evade the current classifier. Similarly, for approaches based on probabilistic models [16][48], the adversary can also easily construct a hard-to-predict attack instance based on cracked knowledge.

MCSs [13][14][15], as the name suggests, uses multiple classifiers rather than only one to improve classifier's robustness. The adversary who wants to effectively evade the classification has to fight with more than one classifier. Although MCSs actually increases the workload of classifiers cracking, it doesn't improve the security of client-side classifiers fundamentally.

In [30], the method *weight evenness* via feature selection optimization is proposed. By appropriate feature selection, the weight of every feature is evenly distributed, thus the adversary has to manipulate a larger number of features to evade detection. In [36], the features are reweighted inversely proportional to their corresponding importance, making it difficult for the adversary to exploit the features. However, given sufficient knowledge, the adversary can easily find enough exploitable features. Besides, in many cases, the adversary can hide the manipulation very deeply without attracting the attention. For example, a phisher can leverage various HTML techniques to make good features invisible.

These defense techniques are built on the assumption that the classification model is kept confidential to the adversary or can be updated timely. However, when the adversary learned sufficient knowledge by cracking classifiers, they can easily and quickly construct effective evasion attacks targeted to the defense techniques.

8. CONCLUSIONS

In this paper, we presented a new attack methodology, *classifier cracking*, for evading the client-side classifier. Our approach is different from existing attack methods. We leverage various reverse engineering techniques to directly extract desirable knowledge from client-side classifier for launching evasion attacks. Our study took GPPF, a learning-based filter for phishing pages deployed in *Chrome* as a case to study, which owns over one billion users. Employing various reverse engineering techniques, we successfully cracked the GPPF model and completely recovered 84.8% encrypted scoring rules. Based on the information, we developed two kinds of evasion attacks: *good features insertion* and *bad features elimination*. The latest 100 real phishing pages collected from PhishTank were taken as the targets of evaluation. The attack experiments showed that we can easily manipulate all the phishing pages (100%) to make them successfully evade the detection of GPPF in the latest version of *Chrome*. Additionally, we analyzed the protection methods that can be potentially applied to client-side classifiers, but regretted to find that it is difficult to thoroughly prevent the client-side classifiers from being cracked using the present technology.

Our research revealed an important fact that the client-side classifiers have a larger attack surface and hence suffer from a larger number of potential attacks. In the future, we will further research potential defense techniques to develop more robust client-side classifier framework.

9. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful comments. The work is supported by National Natural Science Foundation of China (NSFC) under grants 61170240, 91418206 and 61472429, and National Science and Technology Major Project of China under grant 2012ZX01039-004.

10. REFERENCES

[1] Brute Force Attack. https://en.wikipedia.org/wiki/Brute-force_attack

[2] Google has over a billion users of Android, Chrome, YouTube, and search. http://www.theverge.com/2015/5/28/8676599/google-io-2015-vital-statistics

[3] PhishTank. https://www.phishtank.com/

[4] Phishing Attack Trends Report of the 4th Quarter in 2014. http://docs.apwg.org/reports/apwg_trends_report_q4_2014.pdf

[5] Design Documents of Safe Browsing. http://www.Chromium.org/developers/design-documents/safebrowsing

[6] Market share of popular web browsers from Aug 2014 to Aug 2015. http://gs.statcounter.com/#browser-ww-monthly-201408-201508

[7] Google's Safe Browsing service protects 1 billion Chrome, Firefox, and Safari users from malware and phishing. http://thenextweb.com/google/2013/06/25/googles-safe-browsing-service-now-protects-1-billion-Chrome-firefox-and-safari-users-from-malware-and-phishing/

[8] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. D. Spyropoulos, and P. Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. In *Proceedings of the 4th PKDD's Workshop on Machine Learning and Textual Information Access*. 2000.

[9] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure?. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*. ASIACCS'2006. ACM, 16-25.

[10] M. Barreno, B. Nelson, A. Joseph, and J. Tygar. The security of machine learning. *Machine Learning*. 2010. Springer, 81(2): 121-148.

[11] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. *USENIX Annual Technical Conference*, *FREENIX Track*. 2005. 41-46.

[12] B. Biggio, G. Fumera, and F. Roli. Adversarial pattern classification using multiple classifiers and randomization. In *Proceedings of the 2008 Joint IAPR International Workshop on Structural and Syntactic Pattern Recognition*. SSPR'2008. Springer, 5342: 500-509.

[13] B. Biggio, G. Fumera, and F. Roli. Multiple classifier systems for adversarial classification tasks. In *Proceedings of the 8th International Workshop on Multiple Classifier Systems*. MCS'2009. Springer, 5519: 132-141.

[14] B. Biggio, G. Fumera, and F. Roli. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics*. 2010. Springer, 1(1-4): 27-41.

[15] B. Biggio, G. Fumera, and F. Roli. Multiple classifier systems under attack. In *Proceedings of the 9th International Workshop on Multiple Classifier Systems*. MCS'2010. Springer, 5997: 74-83.

[16] B. Biggio, G. Fumera, and F. Roli. Design of robust classifiers for adversarial environments. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. SMC'2011. IEEE, 977-982.

[17] B. Biggio, G. Fumera, F. Roli, and L. Didaci. Poisoning adaptive biometric systems. In *Proceedings of the 2012 Joint IAPR International Workshop on Structural and Syntactic Pattern Recognition*. SSPR'2012. Springer, 7626: 417-425.

[18] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacintoet , and F. Roli. Evasion attacks against machine learning at test time. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. ECML-PKDD'2013. Springer, 8190: 387-402.

[19] B. Biggio, L. Didaci, G. Fumera, and F. Roli. Poisoning attacks to compromise face templates. In *Proceedings of the 6th IAPR International Conference on Biometrics*. ICB'2013. IEEE, 1-7.

[20] B. Biggio, I. Pillai, S. R. Bulò, D. Ariu, M. Pelillo, and F. Roli. Is data clustering in adversarial settings secure?. In *Proceedings of the 6th ACM Workshop on Artificial Intelligence and Security*. AISec'2013. ACM, 87-98.

[21] B. Biggio, G. Fumera, and F. Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*. TKDE'2014. IEEE, 26(4): 984-996.

[22] M. Brü ckner and T. Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD'2011. ACM, 547-555.

[23] M. Brü ckner, C. Kanzow, and T. Scheffer. Static prediction games for adversarial learning problems. *The Journal of Machine Learning Research.* JMLR'2012. MIT Press, 13(1): 2617-2654.

[24] J. Caballero, H. Yin, Z. Liang, and D. Song. Polyglot: Automatic Extraction of Protocol Message Format using Dynamic Binary Analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CSS'2007. ACM, 317-329.

[25] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web*. WWW'2011. ACM, 197-206.

[26] Y. Cao, W. Han, and Y. Le. Anti-phishing based on automated individual white-list. In *Proceedings of the 4th ACM Workshop on Digital Identity Management*. DIM'2008. ACM, 51-60.

[27] N. Chou, R. Ledesma, Y. Teraguchi, and J. C. Mitchell. Client-side defense against web-based identity theft. In *Proceedings of the Network and Distributed System Security Symposium*. NDSS'2004.

[28] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD'2004. ACM, 99-108.

[29] A. Y. Fu, W. Liu, and X. Deng. Detecting phishing web pages with visual similarity assessment based on earth mover's distance (EMD). *IEEE Transactions on Dependable and Secure Computing*. TDSC'2006. IEEE, 3(4): 301-311.

[30] A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd International Conference on Machine Learning.* ICML'2006. ACM, 353-360.

[31] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of the Network and Distributed System Security Symposium.* NDSS'2008.

[32] L. Huang, A. D. Joseph, B. Nelson, B. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Artificial Intelligence and Security*. AISec'2011. ACM, 43-58.

[33] M. Khonji, Y. Iraqi, and A. Jones. Phishing detection: a literature survey. *Communications Surveys & Tutorials*. 2013. IEEE, 15(4): 2091-2121.

[34] P. Kloft and M. Laskov. A "poisoning" attack against online anomaly detection. In *Proceedings of Neural Information Processing Systems (NIPS) Workshop on Machine Learning in Adversarial Environments for Computer Security*. 2007.

[35] L. R. Knudsen, X. Lai, and B. Preneel. Attacks on Fast Double Block Length Hash Functions. *Journal of Cryptology*. 1998. Springer, 11(1): 59-72.

[36] A. Kołcz and C. H. Teo. Feature weighting for improved classifier robustness. In *Proceedings of the 6th Conference on Email and Anti-Spam*. CEAS'2009.

[37] P. Likarish, D. Dunbar, and T. E. Hansen. B-apt: Bayesian anti-phishing toolbar. In *Proceedings of IEEE International Conference on Communications*. ICC'2008. IEEE, 1745-1749.

[38] O. Linda, T. Vollmer, and M. Manic. Neural network based intrusion detection system for critical infrastructures. In *Proceedings of the 2009 International Joint Conference on Neural Networks*. IJCNN'2009. IEEE, 1827-1834.

[39] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of the 2nd Conference on Email and Anti-Spam*. CEAS'2005.

[40] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD'2005. ACM, 641-647.

[41] D. Maiorca, I. Corona, and G. Giacinto. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection. In *Proceedings of the 2013 ACM Symposium on Information, Computer and Communications Security*. ASIACCS'2013. ACM, 119-130.

[42] R. Morris, and K. Thompson. Password security: A case history. *Communications of the ACM*. 1979. ACM, 22(11): 594-597.

[43] A. Narayanan, and V. Shmatikov. Fast Dictionary Attacks on Passwords Using TimeSpace Tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*. CCS'2005. ACM, 364-372.

[44] B. Nelson, M. Barreno, F.J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia.

Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats*. LEET'2008. USENIX, 1-9.

[45] P. Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Proceedings of 23rd Annual International Cryptology Conference*. CRYPTO'2003. Springer, 2729: 617-630.

[46] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta. Phishnet: predictive blacklisting to detect phishing attacks. In *Proceedings of the 29th Conference on Information Communications*. INFOCOM'2010. IEEE, 1-5.

[47] M. A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos. CAMP: Content-agnostic malware protection. In *Proceedings of the Network and Distributed System Security Symposium.* NDSS'2013.

[48] R. N. Rodrigues, L. L. Ling, and V. Govindaraju. Robustness of multimodal biometric fusion methods against spoof attacks. *Journal of Visual Languages & Computing*. 2009. Elsevier, 20(3): 169-179.

[49] B. I. P. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. Lau, S. Rao, N. Taft, and J. D. Tygar. Stealthy poisoning attacks on PCA-based anomaly detectors. *ACM SIGMETRICS Performance Evaluation Review*. 2009. ACM, 37(2): 73-74.

[50] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1983. McGraw-Hill.

[51] A. Slowinska, T. Stancescu, and H. Bos. Howard: A Dynamic Excavator for Reverse Engineering Data Structures. In *Proceedings of the Network and Distributed System Security Symposium*. NDSS'2011.

[52] N. Šrndić and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. SP'2014. IEEE, 197-211.

[53] F. Toolan and J. Carthy. Phishing detection using classifier ensembles. *eCrime Researchers Summit*. eCRIME'2009. IEEE, 1-9.

[54] K. Tretyakov. Machine learning techniques in spam filtering. *Data Mining Problem-oriented Seminar*. 2004. MTAT, 60-79.

[55] S. Udupa, S. Debray, and M. Madou. Deobfuscation: Reverse engineering obfuscated code. In *Proceedings of the 12th Working Conference on Reverse Engineering*. WCRE'2005. IEEE.

[56] T. Wang, T. Wei, G. Gu, and W. Zou. TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. SP'2010. IEEE, 497-512.

[57] W. Weir, S. Aggarwal, B. D. Medeiros, and B. Glodek. Password Cracking Using Probabilistic Context-Free Grammars. In *Proceedings of the 2009 IEEE Symposium on Security and Privacy*. SP'2009. IEEE, 391-405.

[58] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *Proceedings of the Network and Distributed System Security Symposium.* NDSS'2010.

[59] B. Yadegari, B. Johannesmeyer, B. Whitely, and S. Debray. A generic approach to automatic deobfuscation of executable code. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. SP'2015. IEEE.

[60] Y. Zhang, J. I. Hong, and L. F. Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th International Conference on World Wide Web*. WWW'2007. ACM, 639-648.

[61] H. Zhang, G. Liu, T. Chow, and W. Liu. Textual and visual content based anti-phishing: A bayesian approach. *IEEE Transactions on Neural Networks*. 2011. IEEE, 22(10): 1532-1546.