

Obfuscated Malicious JavaScript Detection by Causal Relations Finding

Ismail Adel AL-Taharwa¹, Ching-Hao Mao¹, Hsin-Kuo Pao¹, Kuo-Ping Wu¹, Christos Faloutsos², Hahn-Ming Lee^{1,3}, Shyi-Ming Chen^{1,4}, Albert B. Jeng^{1,4}

¹Dept. of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan

²Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, USA

³Institute of Information Science, Academia Sinica, Taipei, Taiwan

⁴Dept. of Computer Science, Jinwen University of Science and Technology, Taipei, Taiwan

Email: {d9815802,d9415004,wgb,pao,hmlee,smchen}@mail.ntust.edu.tw, christos@cs.cmu.edu, albertjeng@hotmail.com

Abstract—JavaScript code is often obfuscated; given such code, can we tell whether if it is malicious or benign? We propose *Obfuscating Causal Relations Finding (OCRF)*, which addresses this problem. The contributions are the following: (1) careful feature extraction, using domain knowledge (2) no need for de-obfuscation, since our method can be applied to the obfuscated script directly, (3) combined obfuscation detection, with malicious obfuscated code detection (4) improved detection accuracy and significantly reduced false positives (while the average false positive rate of competitors is between 0.18 and 0.30. Our method decreases it between 0.03 and 0.1). Moreover, our method is easy to be implemented as a plug-in for Web browsers.

Keywords— Obfuscation, JavaScript, Security, Sequence Mining, and Hidden Markov Models

I. INTRODUCTION

Nowadays, development of Web applications becomes more convenient by using client-side scripting languages, e.g., VB script and JavaScript [4]. The exposures of these codes allow unauthorized users to reveal them. A common solution to this problem is to hide the semantics of such codes while preserving their functionality, known as *obfuscation*. For example, a simple “Hello” can be obfuscated as several coded representation, (e.g., “\x48\x65\x6C\x6C\x6F”). Generally, obfuscators use techniques to transform scripts into meaningless or unreadable sequence of characters, such as: identifier reassignment or randomization, block randomization, white space and comment randomization, strings encoding, string splitting and integer obfuscation.

Obfuscated JavaScript codes are extremely hard to be understood and reverse-engineered because the aim behind obfuscation is to hide their programming functional details. Hence such codes are possibly malicious files and hard to be detected. In a very recent announcement, Michael Sutton noticed that at least 18 of 41 anti-virus’ software will flag obfuscated benign JavaScript as malicious attack [14]. Actually, obfuscation makes the problem of malicious JavaScript detection more complicated by introducing new patterns that is hard to be discovered by the traditional

malware detectors [10]. Those patterns are shown in Table 1.

TABLE 1. POSSIBLE JAVASCRIPT PAIRS COMPARISONS

Negative category	Positive category	Situation type
Plain benign	Plain malicious	General situation
Plain benign	Obfuscated malicious	Concerned situation
Obfuscated benign	Plain malicious	Concerned situation
Obfuscated benign	Obfuscated malicious	Concerned situation

In this work, we aim to distinguish obfuscated malicious JavaScript from obfuscated benign JavaScript with no need of additional de-obfuscation efforts. Given a set of labeled obfuscated benign and malicious JavaScript files along with a set of plain benign and malicious JavaScript files, our target is to filter the obfuscated malicious JavaScript files out of such collection. We are interested in this issue because there is considerable number of obfuscating techniques and tools which may be adopted by programs developers and attackers [5] in the same way. This research shows that the obfuscated patterns diverse according to their developers’ intention.

Obfuscated JavaScript files can be either benign or malicious with similar code patterns. However, it is possible to discover the intentions behind it based on their hidden semantics. The developers with benign intention use obfuscation to protect their codes’ copyright, while attackers use it to evade detection of their assaults.

In this study, we investigate the discriminative characteristics of the obfuscated codes based on their semantics to improve the identification of malicious codes. We proposed Obfuscating Causal Relations Finding (OCRF) technique to detect the obfuscated malicious JavaScript codes based on capturing causal relations among extracted JavaScript tokens. Since the obfuscated malicious code can be detected without deobfuscation, the detection process does not need decoding or reverse engineering works.

To evaluate the effectiveness of the proposed technique in distinguishing obfuscated malicious JavaScript from

obfuscated benign JavaScript, we measured its performance by incorporating variant kinds of benign obfuscation techniques such as: Hexa obfuscation, MixUp obfuscation and MD5 obfuscation. The experiment results showed better attack detection coverage comparing with other baseline approaches [11], [10].

```
Opera.postError;}else{face={};var lineNumber=(0x404+4996-0x1788),
document.createElement("x44\x40\x56");var consoleBlock=messageBlog
style.borderTop="\x31\x70\x78\x20\x73\x6f\x6c\x69\x64\x20\x23\x43
style.padding="\x31\x70\x78\x20\x31\x30\x70\x78";messageBlock.style
"\x23\x46\x46\x46\x46\x46\x46";consoleBlock.style.font=
"\x31\x30\x70\x78\x2f\x31\x34\x70\x78\x20\x63\x6f\x75\x72\x69\x65
\x63\x65";consoleBlock.style.position="\x61\x62\x73\x6f\x6c\x75\x22
"\x30";consoleBlock.style.bottom="\x30";consoleBlock.style.width="
zIndex="\x31\x30\x30\x30";function setOverflowMode(){overflowmode=
(a)

<Script>
Var Vg='a06d04937ccdc754e9ebc1c93e37da1309ac8e3c68
746d6c3e0a3c626f64793e3c6468762069643d224469754944223
e783c2f6469763e0a3c8363726970743e0a0a66756e6374696f6
E20696e7365727455524c322875726c297b0a0909766172207265
Var HJN="";
Var q=Vg.slice(38,14236);
For(K=38;K<14236;K+=2)
{
HJN+='%'+Vg.slice(K,K+2);
}
Document.write(unescape(HJN));
</Script>
(b)
```

Figure 1. Comparison between obfuscated JavaScript codes: (a) Obfuscated benign JavaScript code. (b) Obfuscated malicious JavaScript code

We present more details of our work in the following sections. In Section II, we mention the related works. Section III describes the details of the proposed architecture to detect obfuscated malicious script. Experiments and evaluations are presented in Section IV. Those experiments results are discussed in Section V. Finally, we conclude this work in Section VI.

II. RELATED WORKS

In this section, we state the related works of malicious JavaScript detection and obfuscation techniques.

A. Malicious JavaScript Detection

JavaScript is flexible in running which make it emergent to create interactive Web pages. One problem accompanied with JavaScript code is that it is downloaded into Web browsers and executed on-the-fly by an embedded interpreter. It facilitates a lot of contemporary attacks. Many of them are mentioned by David Byrne [1] in one of the Black Hat community briefings. This makes it attractive choice for attackers to implement their assaults and distribute them over the Internet. The most pervasive JavaScript attacks are cross site scripting (XSS), SQL Injection and Drive-by-Download attacks [3]. Comprehensive study of these attacks with respect to the use of JavaScript is provided in [6], [8].

Standard JavaScript security solutions implemented by most available Web browsers are Sandboxing [13] and Security Policies. The former one restricts the execution of JavaScript on the client machine. That is, it does not provide any built-in support for file access [4]. The latter one includes several policies, within which we concern about the Same Origin Policy (SOP) that allows the read/write operations from the same origin, and disallows any access to content

from different origins. Origin means the values of protocol, host, and port number all together [7]. Those solutions have much vulnerability, especially when manipulating distributed applications, e.g., easiness of poisoning Domain Name Server (DNS) cache with bogus data using JavaScript [9]. Also, there are many techniques circumventing SOP using JavaScript codes [8].

B. Obfuscation Techniques

Obfuscation techniques originally created for benign purposes to protect the copyright of the developers. However, it provided a vulnerability to the attackers to evade most of the detection mechanisms such as RegExp, rule-based, syntax-based and signature-based ones by making it impossible to trace the code manually or automatically without execution. Figure 1 shows two samples of obfuscated JavaScript. Likarish [11], [10] first proposed to deobfuscate the scripts to trace it. Later he proposed to use some machine learning techniques on a set of data features (*Specific Features Counting*) to detect obfuscated malicious JavaScript.

Deobfuscation solution will be infeasible, especially when it have to be done dynamically. Also, the specific features solution is effective when collecting the obfuscated JavaScript files from black listed domains. However, in real world environment we are obliged to distinguish between obfuscated malicious JavaScript and obfuscated benign JavaScript, this is exactly what the proposed OCRF method does. This work is motivated by the presence of obfuscated benign JavaScript in the reality [5].

We found that there are six principle obfuscation strategies. Table 2 gives qualitative comparison between the performances of the proposed techniques to detect them.

III. OBFUSCATED MALICIOUS JAVASCRIPT DETECTION SYSTEM

In this section, we describe obfuscated malicious JavaScript detection mechanism based on finding the obfuscating causal relations that can reduce the false positive caused by the presence of obfuscated benign instances. To deal with most situations listed in Table 1, the proposed system consists of two hierarchical mechanisms *Obfuscation Detection Techniques (ODT)* and *Obfuscating Causal Relation Finding (OCRF)*. The ODT is responsible for detecting the existence of obfuscation. Furthermore, the OCRF is used to detect the obfuscated malicious JavaScript when obfuscated benign JavaScript existing.

In ODT, we use two obfuscation detection techniques based on n-gram and specific features distributions that could be automatically plugged-in machine learning classifier to build syntax level discriminative models. In OCRF, we propose a novel model to convert JavaScript files to the corresponding tokens' sequences. After that, the causal relations extracting mechanism based on sequence mining technique is used to find out causal relations (semantics) of malicious JavaScript. The system architecture is shown in Figure 2.

A. Obfuscation Detection Techniques

In order to build a model that learns the obfuscating

TABLE 2. COMPARISON BETWEEN DETECTION TECHNIQUES OF JAVASCRIPT OBFUSCATION STRATEGIES ($\sqrt{}$ means weak, $\sqrt{}\sqrt{}$ means medium, and $\sqrt{}\sqrt{}\sqrt{}$ means strong detection ability).

Technique	Identifier reassignment	Block randomization	Whitespace randomization	String encoding	String splitting	Integer obfuscation
Signature	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$
Letter unigram	$\sqrt{}\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}\sqrt{}$	$\sqrt{}$	$\sqrt{}$
Letter bigram	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}$	$\sqrt{}$
Specific features	$\sqrt{}\sqrt{}$	$\sqrt{}$	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}$	$\sqrt{}$
Sequence mining	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}$	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}\sqrt{}\sqrt{}$	$\sqrt{}\sqrt{}\sqrt{}$

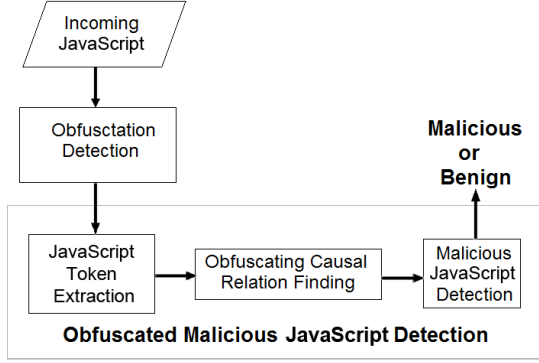


Figure 2. System architecture of proposed method

behavior based on the code patterns, we utilized the specific features work [10] with significant customization to achieve obfuscation detection with minimum overhead. We exploit the idea about specific features with fewer features. Specifically, we used only eight features, which are file length in characters, line count, string count, average string length, encoding count, method invoke counter, average method argument length and percentage of whitespaces. Furthermore we implemented letters unigram and letters bigram techniques, and applied the machine learning methods to build the discriminative models. Also, we proposed the tokens n-gram in similar way to the letters n-gram method.

B. Obfuscating Casual Relation Finding

The second mechanism doing obfuscated malicious JavaScript detection leverages the sequence-based anomaly detection techniques to profile the causal relations of the extracted JavaScript tokens and to distinguish the obfuscated benign from obfuscated malicious JavaScript Web pages. Based on sequence mining and dissimilarity comparing mechanism, we exploited the sequence comparing mechanism developed by Mao et al [12] for identifying obfuscated malicious JavaScript. There are three modules in use to achieve the goal of this mechanism. First, token sequences are extracted from the obfuscated JavaScript sources. Second, OCRF is used to profile semantics of the JavaScript token sequences and to find the intrinsic intentions behind the obfuscation. Finally, classification techniques are applied to build obfuscated malicious JavaScript detection models.

1) Obfuscated Token Extraction: Obfuscated token extraction module can extract the specified tokens from original JavaScript sources. In this work, we define six types of tokens to consider the obfuscated characteristics. The token

types are shown in Table 3.

TABLE 3. TOKEN TYPES FOR DISTINGUISHING THE OBFUSCATED JAVASCRIPT.

Token Type	Description
JS	Refer to JavaScript method call, class or package name, or keyword
HEX	Hexical encoding
UNI	Unicode coding
OCT	Octal coding
SUSP	Refers to those suspicious methods associate with malicious JavaScript scripts, e.g., eval, unescape, replace.
CHAR	JavaScript string related methods, e.g., chatAt, sub-String

2) Obfuscating Causal Relations Finding (OCRF):

Due to the difference between the obfuscation intentions of the benign users and the attackers, OCRF intends to profile the relations of tokens to distinguish between the obfuscated malicious JavaScript and obfuscated benign JavaScript. In OCRF, we applied Markov chain model to profile the token transition relations for each JavaScript token sequence.

Given a JavaScript token sequence e , we look for the Markov chain model (MCM) G that is the most appropriate one to describe the sequence. A MCM $G(V, E, \theta)$ has its vertices V belonging to the event (token) types, while its edges E are the possible transitions between those event types. To be specific, if two events e_t at time t and e_u at time u are directly correlated, a MCM is defined by the model parameters θ recording all transitions, or in $S \times S$ matrix forms the transition matrix A , with $A_{i,j} = P(e_u = j | e_t = i)$, the probability of jumping from state (JavaScript token type) i at time t to state j at time u . The goal of OCRF is to model the correlations between pairs of events. One MCM is built for each sequence.

The log-likelihood $l(e; G)$ of the token sequence $e(t_0 : \cdot)$ given in the model G can be simply written as

$$\begin{aligned}
 l(e; G) &= \log L(e; G) = \log \left(P(e_{t_0}) \prod_{t=t_0+1} P(e_t | e_{t-1}) \right) \\
 &= \log P(e_{t_0}) + \sum_{t=t_0+1} \log P(e_t | e_{t-1}),
 \end{aligned} \tag{1}$$

where L is the likelihood function, the initial probability

$P(e_{t_0})$ is assumed to be constant in this work. Given two sequences of JavaScript tokens e_1 and e_2 and their associated models G_1, G_2 we refer to [12] work to measure the distance between those two sequences which depends on how well one sequence is described by the model for another sequence. Given that the model G is a negative logarithm of the likelihood, we compute the code length of a sequence e as follows:

$$c(e|G) = -l(e; G) = -\log L(e; G). \quad (2)$$

We define the dissimilarity (distance) between two sequences of JavaScript tokens e_1 and e_2 as

$$d(e_1, e_2) = \frac{c(e_1|G_2) + c(e_2|G_1)}{c(e_{12}|G_{12})}, \quad (3)$$

where e_{12} is a new sequence formed by concatenating the sequences e_1 and e_2 . G_{12} is the associated model of the concatenated sequence e_{12} .

Given the distance for each pair of sequences, sequence embedding seeks to represent those sequences of JavaScript tokens in a space so that the distance in the space will represent the relation of those sequences in a proper manner. We adopted Isomap technique [15] as a manifold learning approach before generating dissimilarity matrix to find the causal relation representation of JavaScript token sequences in the coordination spaces.

3) Malicious JavaScript Detecting: It is easy to plug-in the proposed architecture in any classifier to detect anomalies. In this study, we adopted support vector machine (SVM), Naïve Bayes (NB) and Alternating Decision Tree (ADTree) for malicious JavaScript detection according to the extracted features.

IV. EXPERIMENT RESULTS

To evaluate the proposed method, first we collected a sample dataset; then we defined the experimental parameters and conditions in addition to the corresponding appropriate evaluation metrics. This section discusses those two issues in detail, and provides rational evidence on the proposed architecture performance based on the results of the conducted experiments.

A. Dataset

With respect to our objective, we are interested in three categories of JavaScript files: (1) *plain (non-obfuscated) benign*, (2) *obfuscated benign*, and (3) *obfuscated malicious*. Practically we collected plain benign and obfuscated malicious JavaScript files. For the remaining category, we used obfuscation tools to generate it from the plain benign JavaScript; to evaluate the ability of the proposed architecture to cover variant benign JavaScript code obfuscation techniques.

All of the categories have been collected using virtual machine platforms. We used both Microsoft Windows XP and

Ubuntu distribution with disabled antivirus software to facilitate the download of the malicious JavaScript files. In all of our crawling processes we have referred to certain Web sites and domains to validate our corpus. To collect benign samples we referred to the most frequently visited Web sites (first top 100 sites listed in –Alexa Web sites’ ranking page¹. To collect real world malicious JavaScript, we looked on the blacklisted Web pages². Our final corpus consists of 50 obfuscated malicious samples and 206 plain benign samples. Eventually, we formulated three types of datasets using this corpus. These datasets contents are manifested in Table IV.

TABLE 4. PREPARED DATASETS COLLECTIONS

Set ID	Set label	Set content
1	Plain set	Contains obfuscated malicious and plain benign JavaScript files
2	Obfuscated sets	Contain obfuscated malicious and obfuscated benign JavaScript files

For the malicious dataset, it was already obfuscated by the attackers. One interesting characteristic of this set is that it is not based on unique obfuscation algorithm as what is required in [10]. On the other hand, for the benign dataset we maintained it without any modification on the ODT method of this work. However, we applied different kinds of obfuscation techniques using prevalent obfuscating techniques to facilitate the operations in the OCRF of this work.

Several mechanisms of JavaScript obfuscation are available to obfuscate the benign set as shown in Table II. We studied arrangements and methods used in the obfuscation of benign JavaScript. Then we selected the most practical and valid techniques to obfuscate the benign dataset. The main characteristics we seek in the selected tools are: (1) Portability, to ensure the workability of the generated script after the obfuscation process in most of the available web browsers. (2) Reliability, to ensure that the generated script after the obfuscation process satisfies the same intentions of the original source script. Two proofed tools used in this work, one is *Stunrix JavaScript obfuscator* commercial software, and the other is *JavaScript Code Mix Up service* provided by Tools88³ encoding Web site. Also, we incorporated extra encryption technique MD5, even it's not pure obfuscation technique; to provide practical evidence on the resistance of our solution to the extreme complex obfuscation patterns.

B. Evaluation Metrics and Experiment Design

To the best of our knowledge this is the first implementation for obfuscated malicious and obfuscated benign discriminator, which does not need to perform any kind of de-obfuscation preprocessing. We implemented letter ngram, token n-gram and specific features works to detect obfuscated scripts using the plain set described in Table IV. To evaluate the

¹ [URL]: <http://www.alexa.com>

² We referred to the sites listed in the following blacklists:

[URL]: <http://malwareurl.com>

[URL]: <http://malwaredomainlist.com>

³ [URL]: <http://tools88.com/safe/JSencode.php>

performance of the proposed mechanism, we used three different obfuscated sets, those are: Hexadecimal, MixUp and MD5 obfuscating techniques.

The performance measures in all carried experiments are precision, recall, F-measure and error rate [2]. For the first three measures their values were taken directly from Weka toolbox results, but for the last measure it was computed manually with respect to the true set (i.e., in the obfuscation detection stage, obfuscated JavaScript set is the true set). Because of the space limitation, we provided the results of F-measure and error rates only.

To measure the performance of n-gram solution, we wrote the code to extract the n-grams from the JavaScript files and compute the number of occurrences for each gram. Then we normalized these quantities with respect to the corresponding file size (total account of grams in that file). This process is important to eliminate any possible bias that may appear according to the variation in the size of those files. In the state of the specific features, we implemented appropriate procedures to count their occurrences in the JavaScript files, after specifying those desired features.

C. Effectiveness Analysis

Effectiveness of the proposed architecture is evaluated by the enhancement in the detection of JavaScript Obfuscation. Figure 3 shows the results of both F-measure and error rate for detecting obfuscation on the plain dataset using five techniques for comparisons, including: “letter unigram”, “letter bigram”, “specific feature”, “token unigram” and “token bigram”. The letter bigram based models got the best performance with 89% average F-measure over the different classification models, while the token unigram based models were the worst in the prediction with 69% average F-measure value. From the error rate prospective, the letter unigram based models were the best in predicting the obfuscation with 9.2% average error rates, which means it has obfuscation detection ability greater than 90.00%. On the other hand, the worst performance resulted from token unigram based models with 40.6% average error rates.

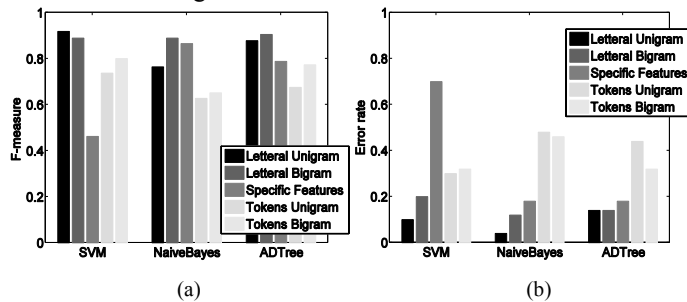


Figure 3. The result of the proposed obfuscation detection techniques over a set of the classification methods (SVM, NB, and ADTree) measured in (a) F-measures and (b) Error rate.

Frankly speaking, the more promising finding of this work is the better detection of the obfuscated malicious JavaScript code. We compared the results of the proposed method with the performance of the specific features method using three obfuscation techniques. Figure 4 shows the results of F-

measure and error rate for these two methods. Figure 4(a) confirms the consistent robustness in the performance of the proposed technique given any obfuscation technique. The average F-measure rate over these experiments was 94%. Also the results of the error rates shown in Figure 4(c) were promising with 0.076 in average. These findings outperform the results using the specific features method significantly with more than 18% accuracy in detection.

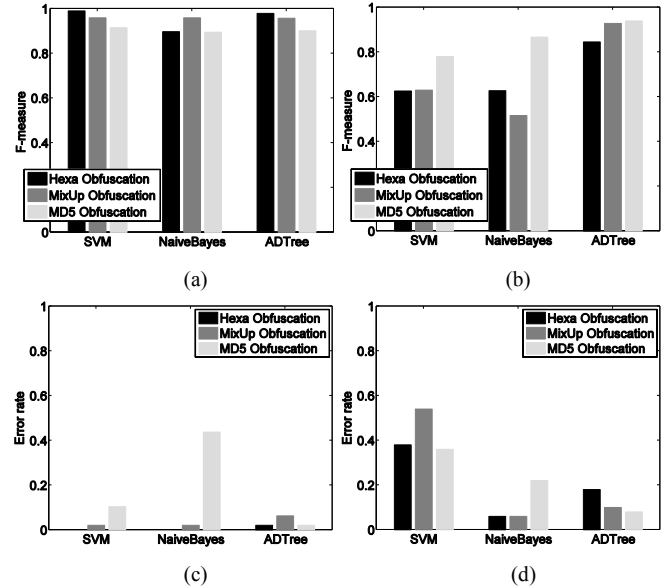


Figure 4. Comparison between the results of our proposed OCRF solution and the specific feature solution in detecting obfuscated malicious JavaScript files over a set of classification methods (SVM, NB, and ADTree). (a) OCRF performance measured in F-measure. (b) Specific features performance measured in F-measure. (c) OCRF performance measured in Error rate. (d) Specific features performance measured in Error rate.

V. DISCUSSION AND ANALYSIS

This section provides detailed discussion about the results of this work from two points of view: (1) Performance of the proposed detection techniques. (2) False positive rates. Figure 3 demonstrates a satisfactory performance of the letters unigram, letters bigram, and tokens bigram methods in detecting obfuscation. However, letters unigram and letters bigram show consistent protection ability independently from the implemented classification model. The most interesting finding is that letters unigram gained the minimum error rates in terms of detecting obfuscation. Also, letters bigram was the best in terms of F-measure metric. Consequently, letters bigram is the best obfuscation detection method in terms of the minimum false positive rate. Figure 4 shows how the proposed OCRF method outperforms the specific features method in detecting obfuscated malicious JavaScript. In addition, it shows the consistent performance of OCRF method given the different obfuscation techniques and using any of the classification models. Moreover, the results of F-measure and error rates presented in Figure 4 proof the minimum false positive rates generated in detecting obfuscated malicious JavaScript which was 0.00% in more than half of these experiments.

VI. CONCLUSION

In this work we have studied the problem of obfuscated JavaScript code, and proposed *Obfuscating Causal Relations Finding (OCRF)* technique to facilitate a better discrimination between obfuscated malicious JavaScript and obfuscated benign JavaScript. Our goal is to detect obfuscated malicious JavaScript with the minimum false positive rates. Our contributions are: (1) careful feature extraction, using domain knowledge (2) no need for de-obfuscation, since our method can be applied directly to the obfuscated script, (3) combined obfuscation detection, as well as malicious (obfuscated) code detection (4) improved accuracy and significantly reduced false positives. (While the average false positive rate of competitors is between 0.18 and 0.30, our method decreases it to be between 0.03 and 0.1. Moreover, our method is easy to implement as a plug-in for a Web browser.

REFERENCES

- [1] D. Byrne. Intranet invasion through anti-dns pinning. Online: <http://www.blackhat.com/html/bh-usa-07/bh-usa-07-speakers.html>, 2007.
- [2] C. Chen, A. Liaw, and L. Breima. Using random forest to learn imbalanced data. Technical report, Department of Statistics, UC Berkeley, 2004.
- [3] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web, WWW '10*, pages 281–290, New York, NY, USA, 2010. ACM.
- [4] O. Hallaraker and G. Vigna. Detecting malicious JavaScript code in mozilla. In *ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, pages 85–94, Washington, DC, USA, 2005. IEEE Computer Society.
- [5] B. Hoffman. Circumventing automated javascript analysis. In *Blackhat 2009 Course Lectures*, April 2009.
- [6] M. P. Jayamsakthi Shanmugam. Cross site scripting-latest developments and solutions: A survey. *International Journal of Open Problems in Computer Science and Mathematics (IJOPCM)*, 1(2):8–28, September 2008.
- [7] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono. On technical security issues in cloud computing. In *Conference on Cloud Computing, 2009. CLOUD '09. IEEE International*, pages 109 – 116, Horst Gortz Inst. for IT Security, Ruhr Univ., Bochum, Germany, Sept 2009.
- [8] M. Johns. On javascript malware and related threats web page based attacks revisited. *Journal in Computer Virology*, 4(3):161–178, August 2008.
- [9] D. Kaminski. Dns server+client cache poisoning, issues with ssl. Online: <http://www.blackhat.com/html/bh-usa-08/bh-usa-08-speakers.html>, august 2008.
- [10] J. E. LIKARISH, P. and I. JO. Obfuscated malicious javascript detection using classification techniques. In *Malware 2009*, Montreal, 2009.
- [11] P. Likarish and E. Jung. A targeted web crawling for building malicious javascript collection. In *DSMM '09: Proceeding of the ACM first international workshop on Data-intensive software management and mining*, pages 23–26, New York, NY, USA, 2009. ACM.
- [12] C.-H. Mao, H.-K. Pao, C. Faloutsos, and H.-M. Lee. SBAD: Sequence based attack detection via sequence comparison. In *ECML/PKDD Workshop on Privacy and Security issues in Data Mining and Machine Learning (PSDML 2010)*, 2010.
- [13] D. S. Peterson, M. Bishop, and R. Pandey. A flexible containment mechanism for executing untrusted code. In *Proceedings of the 11th USENIX Security Symposium*, pages 207–225, Berkeley, CA, USA, 2002. USENIX Association.
- [14] M. Sutton. Antivirus struggling with obfuscated javascript. Online: <http://research.zscaler.com/search/label/obfuscation>, June 2010.
- [15] J. B. Tenenbaum, V. D. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.