Multiple Classifier Systems for More Accurate JavaScript Malware Detection

Zibo Yi [*], *Jun Ma, Lei Luo, Jie Yu,* and *Qingbo Wu*
College of Computer, National University of Defense Technology, China
[*]Corresponding author: Zibo Yi, ziboyi@ubuntukylin.com

Abstract
The researches of JavaScript malware detection focus on machine learning techniques in recent years. These works extract features from JavaScript's abstract syntax tree for the training of classifiers and achieve satisfactory detection results. However, in the training set there exist some scripts that are not so representative and may cause occasional incorrect classification. We propose multiple classifier system (MCS) to reduce this kind of misclassification. As shown in the experiments, the accuracy increases because of the MCS while training time is slightly greater than the original classifier.
Key words: *machine learning, JavaScript malware detection, multiple classifier system*

1 Introduction

Statistical machine learning has been widely used for JavaScript malware detection.[1, 2, 3, 4]These approaches have the common stages. Training stage, using various malicious and benign scripts to train a classifier. Testing stage, classifying scripts to specific classes (benign or malicious). Representative features such as abstract syntax tree (AST), function call patterns are extracted during training and testing stage so that a classifier with satisfiactory accuracy is obtained.

Except for extracting more representative features, some other approaches have been proposed for improving the accuracy. *Zhang et al.*[5] consider that feature selection can drop the insignificant features, thus lets a classifier more accurate and robust. *Perdisci et al.*[6] introduce multiple classifier system for accurate intrusion detection. *Biggio et al.*[7] demonstrate that multiple classifier system (MCS) is effective in security-sensitive applications. These works explore the feasible solutions for improving accuracy in adverse detection applications.

In this paper, we proposed a MCS approach for more accurate JavaScript malware detection with the overhead slightly increased. We use the same training set to train several classifiers and let them vote on a script's class. This approach avoids occasionally incorrect classification. Therefore, the overall accuracy is improved by MCS.

2 JavaScript Malware Classification

In this section, the general procedure of JavaScript malware classification is described. Then the accuracy we want to optimize is defined.

The abstract syntax tree (AST) is often used in source code analysis,[2, 3, 8] which is the first step of training and testing. To illustrate how to utilize AST, we give the following example. Fig. 1 shows a real-world drive-by download attack. The attacker exploits ActiveX to download and run malware (The ActiveX object Wscript.Shell has the privilege of system level operation). We use ASTParser[9] to extract the AST of this code fragment. After that the

AST node is recorded as the feature, which consists of two parts, the node's structure and its content. Here are some feature examples: {FunctionInvocation, XMLHttpDownload}, {StringLiteral, “Wscript.Shell”}, {FunctionDeclaration, f}.

```
function f(path) {
  var url=“http://xxx.net/malware.exe”;
  var data=XMLHttpDownload(url);
  AD2BDStreamSave(path, data);
  var WSH = new ActiveXObject(“Wscript.Shell”);
  WSH.Run(path);
  return 1;
}
```

Fig. 1 – A real-world drive-by download attack

Each script is represented by a feature vector $\boldsymbol{x}=(\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_d)$ after feature extraction. $\boldsymbol{x}_i$ is the existence of corresponding feature. Each script has a label y which indicates the script is malicious or benign (y equals +1 if the script is malicious, -1 otherwise). A classifier c takes $\boldsymbol{x}$ as input and gives the classification result (+1 or -1) as output. During training stage, a classifier c is obtained using machine learning algorithm on training set $T_0=\left\{\left(\boldsymbol{x}^{(i)},y^{(i)}\right)\right\}_{i=1}^{N_0}$. N_0 is the size of training set. During testing stage, the classifier c determines each script in test set $T=\left\{\left(\boldsymbol{x}^{(i)},y^{(i)}\right)\right\}_{i=1}^{N}$ is malicious or not. The accuracy $A(c)$ is the rate of correct classifications. $\frac{1}{2}\left|c\left(\boldsymbol{x}^{(i)}\right)+y^{(i)}\right|$ equals 1 if c classifies $\boldsymbol{x}^{(i)}$ correctly (0 if otherwise). Then,

$$A(c)=\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left|c\left(\boldsymbol{x}^{(i)}\right)+y^{(i)}\right| \tag{1}$$

3 Multiple Classifier System

To optimize the accuracy, we propose the multiple classifier system in this section. Due to the consideration that some ambiguous training samples may lead to incorrect classifications in testing stage, selecting partial samples to train the classifier is feasible to avoid this kind of misclassification. To let the classification result more accurate, several classifiers are trained by the original training set's subsets. Then the class of a script is decided by these classifiers’ votes.

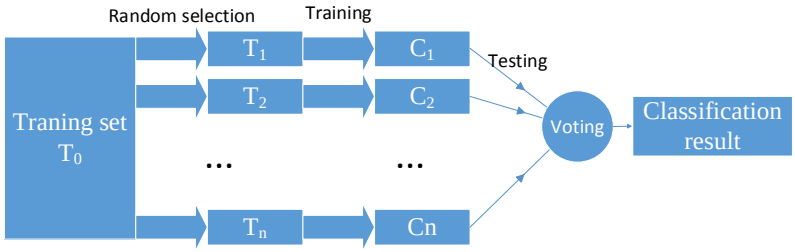

Fig. 2 – A multiple classifier system

Fig. 2 shows the structure of multiple classifier system. The original training set is T_0. T_i is consisted of the elements randomly selected from T_0. The size of T_i is a given percentage (we call it selection rate in the rest of the paper) of the original training set's size. Then a classifier c_i is trained by running a machine learning algorithm L on T_i. After that, these classifiers, which are usually odd, vote on the classification of each script in the test set T.

4 Experiments

4.1 Dataset

We followed the steps described by *Likarish et al.*[10] to collect the JavaScript dataset. We use chrome extension API "devtools.network"[11] to extract JavaScripts and send it to the backend when surfing the Internet. The backend is an HTTP service which receives the POST XMLHttpRequest sent by chrome extension. This can help gathering the benign JavaScript when we browse the website listed in Alexa top 500.[12] As for the malicious JavaScripts, hpHosts[13] collects various types of malicious websites and we browse them. The scripts from these websites are further detected to confirm they're malicious. We use VirusTotal public API[14] to complete the detection automatically. There are 612 malicious and 555 benign scripts collected in total.

After all of the scripts are collected, the AST of each script are extracted by ASTParser[9] then represented as JavaScriptUnit.[15] The AST nodes in JavaScriptUnit are recorded as the features. Each script's feature vector is $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_d)$. $\boldsymbol{x}_d$ equals 0 or 1 which stands for the existence of corresponding feature (see Section 2). The collected scripts' feature vector and their labels consist the dataset $D = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{1167}$.

4.2 Accuracy

The original classifier's training set T_0 is 70% out of D while the remaining consists the test set T. We use scikit-learn,[16] the widely-used machine learning toolkit, to train classifier on T_0 and test the accuracy (defined in Eq. 1) on T.

Table 1 – The accuracy of MCSs and original classifiers

Machine Learning Algorithm	MCS or Original	Selection Rate				
		0.5	0.6	0.7	0.8	0.9
Linear Regression	MCS	0.8689	0.8746	0.8917	0.8831	0.8888
	Original	0.8575	0.8547	0.8774	0.8803	0.8774
Logistic Regression	MCS	0.8632	0.8917	0.8917	0.8746	0.8774
	Original	0.8518	0.8774	0.8945	0.8717	0.8803
Decision Tree Classifier	MCS	0.8632	0.8831	0.9031	0.8774	0.8518
	Original	0.8547	0.8917	0.8917	0.8746	0.8404
Perceptron	MCS	0.8632	0.8689	0.8888	0.8888	0.8490
	Original	0.8490	0.8547	0.8689	0.8831	0.8575

As described in Section 3, we craft the MCS with different selection rate and 4 machine learning algorithms. Table 1 shows the accuracy of these MCSs. We compare MCS with its

original classifier in the case of various selection rates and machine learning algorithms. Most of the time, the accuracy is improved when MCS is used. There are few exceptions mainly because of the random selection of training subset T_i . It should be noted that the accuracy of original classifier with same machine learning algorithm is different in various selection rate. The reason is that we repeat the random partition of dataset D to training set T_0 and test set T every time we train the original classifier.

4.3 Time Overhead

The training time of MCSs and original classifiers is shown by Fig. 3, in which the solid lines represent the MCSs while the same color dotted line represent their corresponding original classifiers. Except for the case of linear regression, the performances of the others are almost the same. Their training time is about 5.5 seconds when using MCS and the time overhead don't increase when the selection rate getting greater. The time overheads of decision tree, logistic regression, perceptron are about 5 seconds. This validates that the using of MCS causes slight time overhead increase. As for the linear regression, its original overhead is significantly greater than the 3 other classifiers and rises sharply as the selection rate increases. So if took the time overhead into consideration, linear regression is not that suitable for JavaScript malware detection.

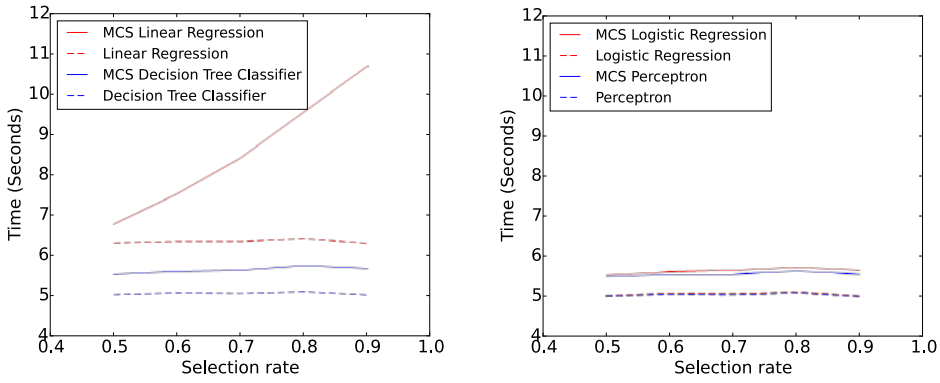

Fig. 3 – MCS and original classifier's time overhead

4 Conclusion

In this paper, we propose the multiple classifier system (MCS), which achieves higher accuracy when we apply it to linear regression, logistic regression, decision tree classifier, and perceptron. Except the case of linear regression, MCS incurs modest overhead: about 10% training time increase. The reason why linear regression incurs great overhead needs to be thoroughly investigated and we leave this to our future work. To conclude, MCS is a proper way to improve the accuracy of JavaScript malware classifier with slight time overhead increase.

Acknowledgements

This work is supported by the NSFC under Grant 61303191, 61303190, 61402504, 61103015.

References

1. *G.Canfora, F.Mercaldo, C.A. Visaggio*, Malicious JavaScript Detection by Features Extraction. e-Informatica Software Engineering Journal. 2014;8(1).
2. *C.Curtsinger, B.Livshits, B.G.Zorn, C.Seifert*, ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. InUSENIX Security Symposium 2011 Aug 8 (pp. 33-48).
3. *J.Wang, Y.Xue, Y.Liu, T.H.Tan*, JSDC: A hybrid approach for javascript malware detection and classification. InProceedings of the 10th ACM Symposium on Information, Computer and Communications Security 2015 Apr 14 (pp. 109-120).
4. *W.Xu, F.Zhang, S.Zhu*, JStill: mostly static detection of obfuscated malicious JavaScript code. InProceedings of the third ACM conference on Data and application security and privacy 2013 Feb 18 (pp. 117-128). ACM.
5. *F.Zhang, P.P. Chan, B. Biggio, D.S.Yeung, F.Roli*, Adversarial Feature Selection Against Evasion Attacks. IEEE transactions on cybernetics. 2015.
6. *R.Perdisci, D.Ariu, P.Fogla, G. Giacinto, W. Lee*,McPAD: A multiple classifier system for accurate payload-based anomaly detection. Computer networks. 2009 Apr 23;53(6):864-81.
7. *B.Biggio, G.Fumera, F.Roli*, Multiple classifier systems for robust classifier design in adversarial environments. International Journal of Machine Learning and Cybernetics. 2010 Dec 1;1(1-4):27-41.
8. *A.Kapravelos, Y.Shoshitaishvili, M.Cova, C.Kruegel, G.Vigna*, Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. In USENIX Security 2013 Aug 14 (pp. 637-652).
9. ASTParser: http://help.eclipse.org/juno/topic/org.eclipse.wst.jsdt.doc/reference/api/org/eclipse/wst/jsdt/core/dom/ASTParser.html
10. *P.Likarish, E.Jung*, A targeted web crawling for building malicious javascript collection. InProceedings of the ACM first international workshop on Data-intensive software management and mining 2009 Nov 6 (pp. 23-26). ACM.
11. Google chrome devtools.network API, https://developer.chrome.com/extensions/devtools.network
12. Alexa topsites, http://www.alexa.com/topsites
13. hpHosts database, http://hosts-file.net/?s=Browse
14. VirusTotal public API, https://www.virustotal.com/en/documentation/public-api/
15. JavaScriptUnit, http://help.eclipse.org/juno/topic/org.eclipse.wst.jsdt.doc/reference/api/org/eclipse/wst/jsdt/core/dom/JavaScriptUnit.html
16. *F.Pedregosa, G.Varoquaux, A.Gramfort, V.Michel, B.Thirion, O.Grisel, M.Blondel, P.Prettenhofer, R.Weiss, V.Dubourg, J.Vanderplas*, Scikit-learn: Machine learning in Python. The Journal of Machine Learning Research. 2011 Feb 1;12:2825-30.

