Detection of Plugin Misuse Drive-By Download Attacks Using Kernel Machines

Manoj Cherukuri
Computer Science
Institute for Complex Additive and System Analysis
New Mexico Institute of Mining and Technology
Socorro, NM, USA
manoj@cs.nmt.edu

Srinivas Mukkamala
CAaNES LLC.
Institute for Complex Additive and System Analysis
New Mexico Institute of Mining and Technology
Socorro, NM, USA
srinivas@cs.nmt.edu

Dongwan Shin
Computer Science
New Mexico Institute of Mining and Technology
Socorro, NM, USA
doshin@cs.nmt.edu

***Abstract* – Malware distribution using drive-by download attacks has become the most prominent threat for organizations and individuals. Compromised web services and web applications hosted on the cloud act as the delivery medium for the exploits. The exploits included often target the vulnerabilities within the plugins of the web browsers. Implementing security controls to counter the exploits within the browsers for ensuring end point security has become a challenge.**

In this paper, a set of features is proposed and is extracted by monitoring the communications between the browser and the plugins during the rendering of webpages. The Support Vector Machines are trained using the defined features and the performance of the trained classifier is evaluated using a dataset with both malicious and benign use cases of the plugins. The dataset included 10,239 malicious use cases and 37,369 benign use cases. To compensate the imbalance in the distribution of the dataset, experiments were performed using weighted costs and oversampling. Our analysis shows that the Support Vector Machines trained by using the proposed set of features classified with an average accuracy of about 99.4%. On integrating the proposed approach as an inline defense, an average performance overhead of 5.14% was observed.

Keywords- plugin exploits; drive-by download; web malware;

I. INTRODUCTION

The implementation of defense in depth strategy for securing large networks and individual machines at network level had shifted focus of the attack vectors on to the port 80. The traditional HTTP (Hypertext Transfer Protocol) attacks like webpage defacements are being monetized by the attackers using the drive-by download attacks. Attackers are hosting the watering hole attacks by compromising legitimate web applications and serving malicious code to users requesting the compromised webpages or web services. Attackers exploit vulnerabilities like Cross Site Scripting (XSS), Uniform Resource Locator (URL) redirection, Injection, Response Splitting to compromise and redirect the traffic towards webpages hosting the malicious code.

The drive-by download attacks often involve multiple types of webpages namely, landing, exploit, distribution and intermediary webpages. The landing webpage is often the compromised webpage that generates the traffic to the exploit webpages using a redirection or a frame. The exploit webpages fingerprint the environment of the machine generating the request and responds with the code containing a targeted attack. Large numbers of landing webpages share a common exploit webpage. The distribution webpages are the webpages used in malware distribution for hosting malware which are downloaded and installed on to the victims machines post exploitation. The intermediary webpages act as traffic redirectors making the attack network complex and untraceable. Except for the landing page, all the other webpages are under the control of the attackers.

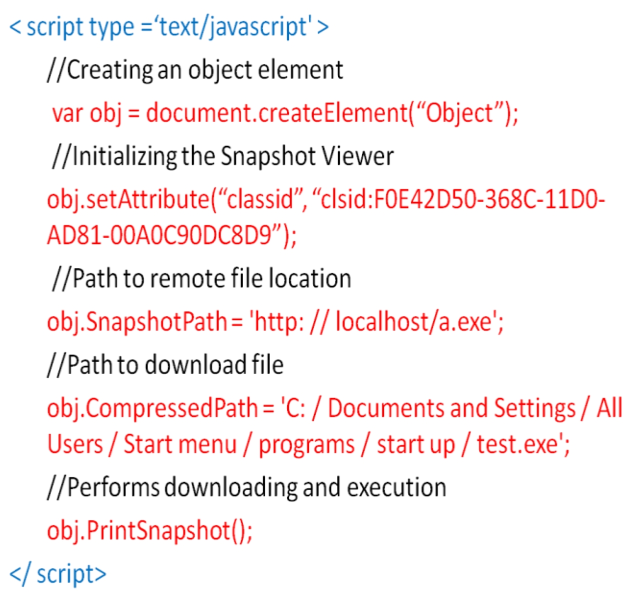
With web browsers acting as the single point of access for diverse web applications and web services hosted on the cloud environments, the attacks in the exploit webpages target the vulnerabilities associated with the browsers. The malicious code fingerprints the browser environment from the request headers and also by using the asynchronous communications with the browser.

The attacks are often identified to target exploits in the plugins installed on the browsers. As the plugins are developed across the globe by different teams, it is challenging to detect the bugs that could be exploited to perform drive-by download attacks. Over 300 browser plugin

COLLABORATECOM 2014, October 22-25, Miami, United States
Copyright © 2014 ICST
DOI 10.4108/icst.collaboratecom.2014.257749

vulnerabilities were identified in years 2010 and 2011, and the trend continues [1].

The drive-by attack targeting the vulnerability in Microsoft's Snapshot Viewer control is shown in Fig. 1. This control is used for viewing the Microsoft Access files within the browser. To view using the Snapshot Viewer, the control needs to be initialized followed by setting the path of the file to be viewed. Snapshot Viewer is set to download the file to the temporary files folder. The vulnerability provided access to the function for setting the location where the file needs to be stored.



```
< script type ='text/javascript' >
    //Creating an object element
     var obj = document.createElement("Object");
     //Initializing the Snapshot Viewer
    obj.setAttribute("classid", "clsid:F0E42D50-368C-11D0-
    AD81-00A0C90DC8D9");
     //Path to remote file location
    obj.SnapshotPath = 'http: // localhost/a.exe';
    //Path to download file
    obj.CompressedPath = 'C: / Documents and Settings / All
    Users / Start menu / programs / start up / test.exe';
    //Performs downloading and execution
    obj.PrintSnapshot();
</ script>
```

Figure 1. JavaScript code exploiting the vulnerability in Microsoft's Snapshot Viewer

In the exploit code shown in Fig. 1, adversaries point the remote file location to a malicious executable and the local path for storage is set to the startup folder with the help of "CompressedPath" method, to ensure its execution. The "PrintSnapshot" method is called for performing the drive-by attack.

As the legitimate web applications and services are compromised, the trust of the user on the legitimate websites is exploited and the traffic is redirected towards malicious websites. Therefore implementing the security controls within the browsers to ensure end point security has become necessary. For protecting the clients from such attacks, a machine learning technique for the detection of the malicious code exploiting the plugin vulnerabilities was proposed and evaluated in this paper.

II. RELATED WORK

The drive-by downloads targeting the vulnerabilities in the web clients are on the rise. The security issues have prompted researchers to come up with novel solutions for countering the attacks. Several works have been proposed in this direction using different types of approaches.

Similar to the traditional honeypots which were used to collect exploits and malware, systems are proposed to crawl and visit webpages using specially built machines for detecting the client side attacks. These systems are referred as the honeyclients. High interaction honeyclients are heavy weight systems that visit webpages in a virtual machine and detects the attacks based on the system state changes and other events [2][3][4]. The high interaction honeyclients take large amount of time and hardware resources and are not scalable to the current size of the Internet. The attacks are detected in these systems only if a download happens.

As an alternative, the researchers have proposed low-interaction honeyclients that would rely on emulation based techniques for detecting the attacks. The low interaction honeyclients are much faster than the high interaction honeyclients and are scalable. PhoneyC [5] uses emulated environment to visit webpages and compares the code identified with the predefined attack signature patterns for the detection of the attacks. Wepawet [6] uses an emulated environment for the execution of the code in the webpages and uses anomaly based model for the detection of the malicious code based on the predefined set of features. These approaches are prone to be evaded by the modern attacks which include fingerprinting code to detect emulated and virtual environments. This limitation has made researchers come up with techniques for handling the evasion techniques in such detection approaches [7] [8]. However, the proposed approach could be integrated into the web browsers and would not be prone to such evasion techniques.

WebWinnow [9] uses the defense and attack mechanisms employed in the exploit kits for the detection of the drive-by attacks. Features related to the behavior, redirections, obfuscation and cloaking are used for the analysis. Different machine learning models were evaluated using the webpages from the exploit kits and the webpages from the wild. Detection of shellcodes in the drive-by download attacks using the kernel machines with the help of opcodes frequencies was proposed [10]. As the presence of shellcodes in webpages is malicious, it is effective in detecting only the attacks that use shellcodes for performing the drive-by downloads.

Zozzle [11] is a lightweight solution based on static analysis for the detection of JavaScript malware. Zozzle is designed for integration into the browser and relies on the construction of the Abstract Syntax Tree (AST) through static parsing of the JavaScript code and uses Naïve Bayesian classifier to detect the attacks. BrowserGuard [12] proposes a behavior based approach for detecting the drive-by download attacks. The fundamental idea behind this approach is to identify the files that are downloaded without the consent of the user and prevent such files from execution.

Nozzle [13] proposed a mechanism for the detection of heap spraying attacks based on the analysis of the objects allocated in the heap and measuring the health metrics of the allocated heap. Nozzle checks each object allocated in the browser's heap for detecting the malicious intent.

In [14], the authors have proposed the detection of ActiveX exploits based on the inter-module communications. The communications between the browser and the plugins are monitored. A vulnerability signature database was constructed from the known set of attacks and symbolic constraint signatures are used for detection. In order to track the attacks involving multiple steps, the session is tracked for each plugin. For the plugins having multiple vulnerabilities, multiple signatures were constructed and validated individually. The authors have proposed to track the session states only for the objects that have an entry in the signature database. Therefore, this approach is limited to the set of vulnerabilities for which the attack signatures are defined. The results demonstrated an overhead of 15%. The overhead introduced is proportional to the signatures in the database and would increase with the vulnerability signatures.

The detection of misuse in the ActiveX controls relying on the graph based model constructed from the functionality of the plugins was presented by the researchers [15]. Each plugin under study is executed with the help of test cases and a model consisting of all the function calls is generated. The function calls that invoke the dangerous APIs (which lead to privileged operations) within the browser are listed. The known plugin misuse attacks are blacklisted for detecting the attacks at runtime. The approach was evaluated by considering three ActiveX controls. Keeping track of the plugins built globally and constructing the models which have complete coverage of the plugins functionality is hard, providing room for the evasion of the attacks. In addition the inclusion of new ActiveX controls into the model would have an adverse impact over the performance.

In contrary to the other approaches where features are more on the evasion techniques and the malicious code including techniques like heap spraying and signature based, the set of features has been proposed to focus on the method calls, the arguments and the parameters initialized with respect to the plugins in legitimate and malicious use cases. The set of features and the discussed mechanism to handle the obfuscations provide efficient and accurate means for detecting the malicious use case scenarios of the browser plugins.

In this paper, we propose to identify the misuse of the functionalities provided by the plugins by building a learning model using the proposed set of features. The proposed approach has been integrated into the browser and the overhead in the performance was also studied.

III. SYSTEM

The system uses the proposed feature set that is extracted from the benign and malicious use cases of different plugins to generate a model using the Support Vector Machines. The generated model would have the capability to judge between the malicious and benign use cases. The features proposed are both specific to the plugins and generic, enabling the approach to be extensible to the newer or unseen plugins. The challenging job in handling the attacks is to overcome the obfuscations performed using client-side interpreters like JavaScript. To overcome the limitations of the obfuscations, the system is designed to read the communications between the plugins and the browser.

A. Plugin Calls Tracker

The malicious code often uses client-side scripting languages to obfuscate the code and hide the malicious intent. The obfuscations are employed using a wide range of techniques making it hard to de-obfuscate and perform the analysis. The obfuscations evade the anti-virus signatures and the other static detection mechanisms in place. In our system, the communication between the plugins and the browser is leveraged to track the plugin calls and the parameters passed. Netscape Plugin Application Programming Interface (NPAPI) is a cross-platform plugin-architecture that's widely in use [16]. We explain the proposed approach using the implementation of NPAPI in the modern web browsers like Firefox.

On initializing the plugin, the browser and the plugin exchange the address of the different functions on both the ends to establish the communication between them. Fig. 2 represents the communication of the web browsers with the plugins. All the functions with the name "NPP_*" are provided by the plugin to the browser and the functions with the name "NPN_*" are provided by the browser to the plugin. Further communication between the plugins and the browser is established through this set of functions. In order to handle the obfuscations and monitor the plugin function call, a hook is placed on these functions.

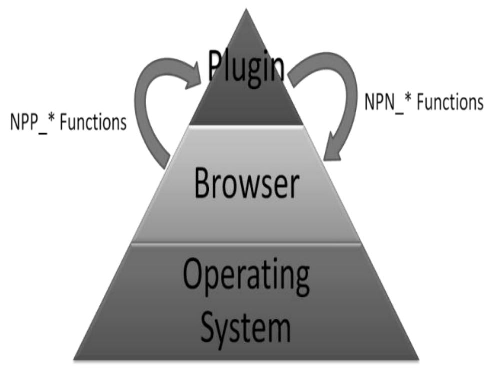


Figure 2. NPAPI IMPLEMENTATION OF PLUGIN AND BROWSER COMMUNICATIONS

B. Features

The proposed system intercepts the function calls made to the plugins and extracts the features from the calls which support our learning machine to classify between a malicious use case and a benign use case. These features are learnt from labeled dataset and the generated model would be intelligent to detect malicious use cases of the plugins. The proposed set of features relies on the differences that are observed between the normal and malicious activities. The features that have been used in our study are described.

Feature 1: Presence of a method. The methods that are invoked within the plugin are monitored. It helps the learning machine about the sets of methods involved in malicious and legitimate use cases.

Feature 2: Number of times a method is called. The attacks might involve calling the same method repetitively which would deviate from the normal use case of the plugins. This feature keeps track of the frequency of the function calls made by the plugins.

Feature 3: Number of arguments passed to a method. This feature monitors the number of arguments that are most frequently used. It facilitates to detect exploits targeting the method parameters that are exposed unintentionally by the developers.

Feature 4: Argument types in methods at different positions. The argument types at different indices of the method calls are tracked. It helps to identify the exploits targeting vulnerabilities that arise from the type checking failure.

Feature 5: Value of the method argument for numeric types. This feature would keep track of the usage of high memory addresses for hijacking the instruction pointer.

Feature 6: Length of the string argument with respect to method and position. It is often observed in memory overflow attacks that strings with large lengths are utilized in the exploits. This feature keeps track of the length of the strings initialized.

Feature 7: Number of non-printable characters in the string arguments with respect to the method and position. In order to exploit the vulnerabilities within the plugins, the exploit might have to send complex data in the form of arguments. This feature keeps track of the non-printable characters occurring in legitimate and malicious use cases.

Feature 8: Number of printable characters with in the string arguments with respect to the method and position. As in feature 6, this feature keeps track of the printable characters occurring in legitimate and malicious use cases.

Feature 9: Properties Initialized. The properties initialized by the plugins are monitored and this helps to detect attacks relying on unintentionally exposed properties.

Feature 10: Number of times a property is set. The frequency of the initializations of the property is studied to detect anomalies due to malicious intents.

Feature 11: Value type set to the property. The types associated with the properties of the plugins are tracked. It helps to identify the exploits targeting vulnerabilities that arise from the type checking failure.

Feature 12: Value of the property set. This feature monitors the values of the property that has been set. The malicious code often refers to high memory addresses to hijack the instruction pointer. This feature would facilitate the detection of such attacks.

Feature 13: Number of non-printable characters in the string property values. In order to exploit the vulnerabilities within the plugins, the exploit might assign complex data to the properties. This feature keeps track of the non-printable characters occurring in legitimate and malicious use cases.

Feature 14: Number of printable characters in the string property values. In order to exploit the vulnerabilities within the plugins, the exploit might assign complex data to the properties. This feature keeps track of the non-printable characters occurring in legitimate and malicious use cases which facilitates in tracking the arguments with shellcodes or memory addresses.

Feature15: The presence of a plugin and the number of times the plugin is created. This feature keeps track of vulnerabilities exploited by targeting multiple plugins and the attacks which necessitate multiple instantiations of the same plugin.

Feature16: Number of printable and non-printable characters used in the plugin function calls and the properties. This feature tracks the overall usage of the special characters in the plugin lifecycle.

Feature 17: Maximum length of the string argument or the string property used in the lifecycle of the plugin.

Feature 18: Maximum numerical value and the standard deviation among the numerical values. This feature keeps track of the values used by the plugins in malicious and benign use case scenarios.

These features are extracted over the channel used for communications between the browser and the plugins. The extracted features are used to train and evaluate the Support Vector Machines (SVMs) towards the detection of plugin exploit code.

IV. EVALUATION

The dataset constructed contains instances of plugin usages for both benign and malicious purposes. For benign use cases the plugins in about 22,000 webpages listed on Alexa [17] and the plugins in the webpages obtained by performing plugin targeted meta-searches were collected. The proposed implementation for monitoring the plugin calls was included in the HTMLUnit [18], an emulated browser. The HTMLUnit has been modified to simulate the presence of the plugins and the functions accessed by the browser. The proposed set of features was extracted during the visit to the webpages using the custom built browser. The malicious use cases were collected using the exploit kits, Metasploit [19] and malicious webpages from the wild. Our dataset comprised of 37,369 benign use cases and about 10,239malicious use cases of the plugins. As the distribution of the samples between the normal and malicious use cases is not equal, the experiments were performed by altering the cost of misclassification for a malicious use case and generating synthetic data for malicious use cases.

C. Model selection using SVMs

In any predictive learning task, such as classification, both a model and a parameter estimation method should be selected in order to achieve a high level of performance of the learning machine. Recent approaches allow a wide class of models of varying complexity to be chosen. Then the task of learning amounts to selecting the sought-after model of optimal complexity and estimating parameters from training data [20][21]. Within the SVMs approach, usually parameters to be chosen are:

- The penalty term C which determines the trade-off between the complexity of the decision function and the number of training examples misclassified;
- The mapping function φ.
- The kernel function such that

$$K(x_i, x_j) = \varphi(x_i).\varphi(x_j).$$

In the case of RBF kernel, the width, which implicitly defines the high dimensional feature space, is the other parameter to be selected.

Model selection was performed on the training datasets by considering 50% of the normal use cases and 50% of the malicious use cases. Grid search was performed with 5-fold cross validation to select the C and γ values. Grid search verifies the accuracies of the model using the cross-validation for different values of C and γ and returns the C and γ value pair with optimal accuracy.

Grid search was performed over C and γ values ranging from -5 to 15 and -15 to 3 respectively. Fig. 3 shows the grid search plot obtained from the training dataset. Due to the imbalance in the data distribution of malicious and benign instances, the grid search was performed by using weighted costs. Since the ratio of benign to malicious use cases is approximately four by number, the cost of misclassification of malicious use case is set to be four times the cost of misclassification of benign use case. The (C, γ) value pair with optimal detection accuracy was (8192, 0.0000305).

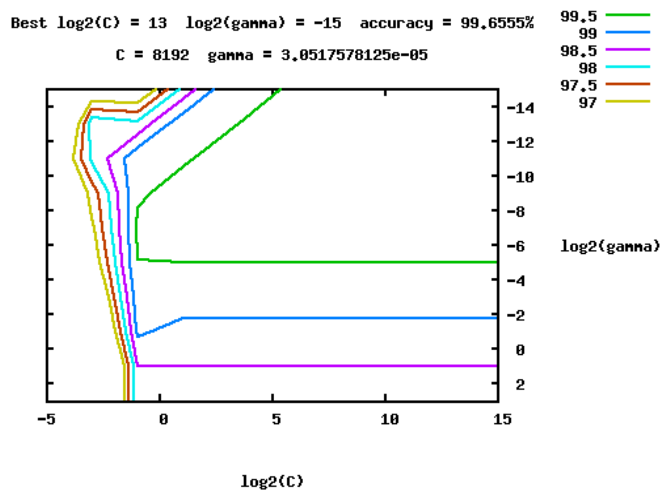


Figure 3. RESULTS OF GRID SEARCH FOR COST(C) AND GAMMA(Γ) VALUES ON TRAINING DATASET

D. ROC Curves

Receiver Operating Characteristic (ROC) curve is the plot of sensitivity on y-axis versus the specificity on x-axis. The ROC curves represent the trade-off between the true positives and false positives. The x-axis represents the false positive rate and the y-axis represents the true positive rate. The point (0, 1) represents an ideal classifier with 100% true positive rate and 0% false positive rate.

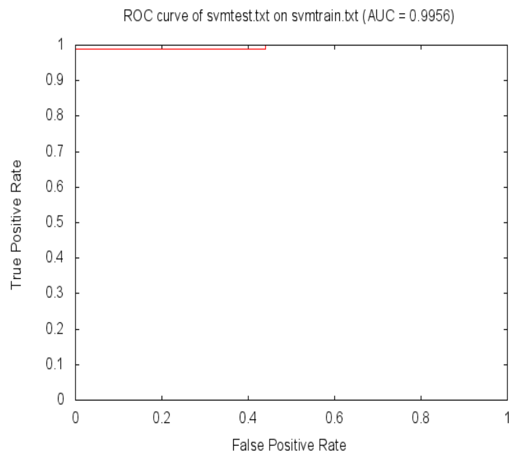


Figure 4. ROC CURVE FOR THE CLASSIFICATION MODEL OBTAINED BY USING TRAINING DATASET 50% OF THE ORIGINAL DATASET

The ROC curves are plotted by considering the training files of 50% of the total number of samples in each class and testing on the remaining samples of the dataset. The ROC curves obtained from the considered dataset using the selected C and γ values is shown in Fig. 4.

The Area Under Curve (AUC) on each plot represents the overall accuracy of the classifier. The AUC under the ROC curve for the model generated by training 50% of the samples was 0.9956.

E. Detection Effectiveness

SVM classifier was built by training with 50% of the total dataset samples and tested on the remaining dataset of samples. The false positive rates, false negative rates and overall detection accuracy obtained by using the generated classifier were measured. The optimal values for C and γ selected through the grid search were used.

The results observed in the evaluation are presented in Table I. The cost for misclassification for malicious instance was set to be four times the cost of misclassification of a benign instance. This was done to compensate the imbalanced distribution of benign and malicious use cases in the dataset. The accuracies observed are listed in Table I under "Weighted Costs" column.

TABLE I. DETECTION EFFECTIVENESS OF TRAINED SVMS

	Weighted Costs	Over Sampling
Accuracy	99.72%	99. 03%
False Positive Rate	0.03%	0.71%
False Negative Rate	1.15%	1.21%

As high false negative rates were observed on using the weighted costs, the classification accuracy was evaluated on balanced data using synthetically generated data from the existing data points. Synthetic Minority Over-Sampling Technique (SMOTE) [22] algorithm was used for the generation of synthetic data for the malicious use cases. In this oversampling technique new data points are created rather than duplicating the existing data points. Each sample from the minority class is considered and a sample is generated along the line joining the *k* minority class nearest neighbors.

The number of nearest neighbors (k) was set to be five, the default value. Based on the amount of over-sampling required, neighbors among the k nearest neighbors are selected randomly. If 100% oversampling is desired, one data point is randomly selected from the 5 nearest neighbors with respect to the original data point and a synthetic data point is created along the line joining the original and the randomly chosen data point. Series of steps involved in generating the synthetic data points:

- Compute the difference between the original data point and the randomly chosen data point
- Multiply the difference with a random number between 0 and 1
- Adding the generated vector to the original data point results in the new synthetic data point along the line joining the original and the randomly chosen data point.

Synthetic data of 30,717 malicious samples were created using the SMOTE technique and the detection accuracy was computed by using 50% of the samples for training and 50% of the samples for testing. The accuracies observed are listed in Table I under "Over Sampling" column.

False positive rates and the false negative rates are calculated using the following equations.

$$FalsePositiveRate = \frac{FalsePositives}{TrueNegatives + FalsePositives}$$

$$FalseNegativeRate = \frac{FalseNegatives}{TruePositives + FalseNegatives}$$

- False positives are the instances where the benign use cases are classified as malicious use cases
- False negatives are the instances where the malicious use cases are classified as benign use cases
- True positives are the instances where the malicious use cases are classified as malicious use cases
- True negatives are the instances where the benign use cases are classified as benign use cases

The proposed approach relying on SVMs classified with an average accuracy of 99.40%. Though the approach resulted in false positives, the false positive rate is minimal with 0.03% and 0.71%. The approach resulted in a false negative rate of 1.15% and 1.21%. The minimum false negative rate of 1.15% was observed on assigning higher weights to misclassification on a malicious instance. Though false negative rate is higher, in applications like web browsers false positive rate has a severe impact on the usability which is small. The model is designed with complete focus only on the methods, arguments and properties associated with the plugins unlike the approaches that include URL based features, IP address based features and obfuscation based features. This makes the approach more generic towards the detection of plugin misuse vulnerabilities.

F. Performance Evaluation

The performance overhead introduced by implementing the proposed mechanism is computed by visiting the top 1500 webpages of Alexa [17]. The implementation is done in an emulated browser, HTMLUnit [18]. The HTMLUnit has been modified to emulate the presence of the plugins and the corresponding methods. For performing the experiment, the clean HTMLUnit and the custom built HTMLUnit with the proposed implementation were used.

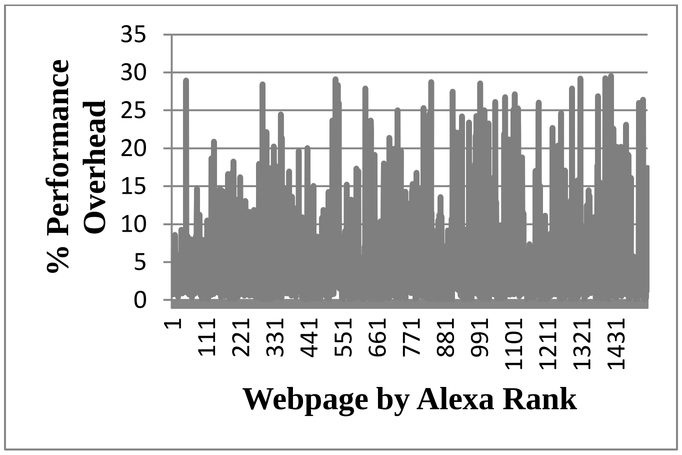


Figure 5. PERFORMANCE OVERHEAD INTRODUCED ON VISITING TOP WEBPAGES

The performance overhead is evaluated by comparing the load times of the webpages. An average performance overhead of 5.14% was introduced by the proposed approach with a standard deviation of 5.57%. Fig. 5 and Fig. 6 show the observed overheads for the top 1500 and top 20 webpages by traffic.

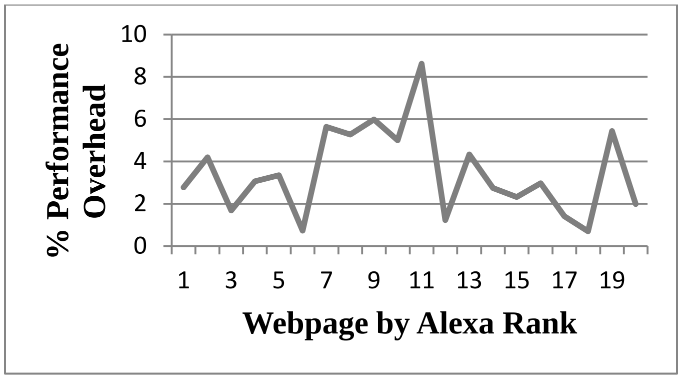


Figure 6. PERFORMANCE OVERHEAD INTRODUCED ON VISITING THE TOP 20 WEBPAGES

V. Conclusions

An approach for the detection of the code exploiting the plugin vulnerabilities using machine learning was proposed and evaluated. The proposed features are generic, not specific to any vulnerability and are completely based on the usage of the plugins in the browsers.

Grid search has been performed to select the optimal values for the cost and gamma parameters. The Support Vector Machines model trained and built using the proposed set of features performed well with the average detection accuracy of about 99.40%.

The model built using the Support Vector Machines learns from both the benign and malicious use cases and selects support vectors maximizing the area of separation. This offers advantages in detection and scalability compared to the signature based detection approaches. The overhead incurred due to the proposed approach was computed and was observed to be 5.14%.

REFERENCES

[1] "Web Browser Plug-in Vulnerabilities", Symantec (2012). Available: http://www.symantec.com/threatreport/topic.jsp?id=vulnerability_trends&aid=web_browser_plug_in_vulnerabilities [13 May 2012].

[2] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A Crawler-based Study of Spyware on the Web. In Proceedings of the 2006 Network and Distributed System Security Symposium, San Diego, CA, pages 17–33, February 2006.

[3] "Capture-HPC Client Honeypot / Honeyclient", TheHoneynet Project (2 Sep 2008). Available: https://projects.honeynet.org/capture-hpc [15 Jan 2012].

[4] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated Web Patrol with Strider HoneyMonkeys. In Proceedings of the 2006 Network and Distributed System Security Symposium, pages 35–49, San Diego, CA, February 2006.

[5] J. Nazario. PhoneyC: a virtual client honeypot. In Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threat, Boston, MA, April 2009.

[6] M. Cova, C. Kruegel, and G. Vigna. Detection and Analysis of Drive-by-download Attacks and Malicious JavascriptCode. In Proceeding of the 19th International World Wide Web Conference, Raleigh, NC, April 2010.

[7] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifer. Rozzle: De-cloaking internet malware. In Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, May 2012.

[8] A. Kapravelos , Y. Shoshitaishvili , M. Cova , C. Kruegel , G. Vigna, Revolver: An Automated Approach to the Detection of Evasive web-based Malware. In Poceedings of the 22nd USENIX conference on Security, Washington, D.C, August 2013.

[9] B. Eshete and V. N. Venkatkrishnan. WebWinnow: Leveraging Exploit Kit Workflows to Detect Malicious Urls. In Proceedings of the 4th ACM conference on Data and Application Security and Privacy, San Antonio, TX, March 2014.

[10] M. Cherukuri, S. Mukkamala, and D. Shin: Detection of Shellcodes in Drive-by Attacks using Kernel Machines. In Journal of Computer Virology and Hacking Techniques, Vol. 10, Issue 3, pages 189-203, August 2014.

[11] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. Zozzle: Low-overhead mostly Static Javascript Malware Detection. In Proceedings of the USENIX Security Symposium, San Francisco, CA, August 2011.

[12] F. Hsu, C. Tso, Y. Yeh, W. Wang, and L. Chen. BrowserGuard: A Behavior Based Solution to Drive-by-Download Attacks. In IEEE Journal on Selected Areas in Communications, Vol. 29, No. 7, pages 1461-1468, August 2011.

[13] P. Ratanaworabhan, B. Livshits, and B. Zorn. Nozzle: A Defense Against Heap-spraying Code Injection Attacks. In Proceedings of the USENIX Security Symposium, Montreal, Canada, August 2009.

[14] C. Song , J. Zhuge , X. Han and Z. Ye. Preventing Drive-by Download via Inter-Module communication monitoring. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, Beijing, China, April 2010.

[15] T. Dai, S. Sathyanarayan, R. H. C. Yap and Z. Liang. Detecting and Preventing ActiveX API-Misuse Vulnerabilities in Internet Explorer. In Proceedings of Information and Communications Security, Vol. 7618, Hong Kong, China, October 2012.

[16] "NPAPI." Wikipedia (17 Jun 2014). Available http://en.wikipedia.org/wiki/NPAPI [19 Jun 2014].

[17] Alexa: Top Sites. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip [11 Sept 2013]

[18] “HtmlUnit”, Gargoyle Software (05 Aug 2010). Available: http://sourceforge.net/projects/htmlunit/files/ [05 Jun 2011].

[19] Metasploit: Rapid7. http://www.metasploit.com. [15 May 2012]

[20] J. H. Lee and C. J. Lin.Automatic Model Selection for Support Vector Machines. Technical Report, Department of Computer Science and Information Engineering, National Taiwan University, 2000.

[21] V. Cherkassy. Model complexity Control and Statistical Learning Theory. In Journal of Natural Computing, Vol. 1, pp. 109-133, 2002.

[22] N. Chawla, K. Bowyer,L. Hall, and W. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. In Journal of Artificial Intelligence Research, Vol. 16, pp. 341-378, 2002.