International Journal of Information Security and Privacy
Volume 11 • Issue 4 • October-December 2017

Detection of Drive-by Download Attacks Using Machine Learning Approach

Monther Aldwairi, Jordan University of Science and Technology, Department of Network Engineering and Security, Irbid, Jordan
Musaab Hasan, Zayed University, College of Technological Innovation, Abu Dhabi, U.A.E
Zayed Balbahaith, Zayed University, College of Technological Innovation, Abu Dhabi, U.A.E

ABSTRACT

Drive-by download refers to attacks that automatically download malwares to user's computer without his knowledge or consent. This type of attack is accomplished by exploiting web browsers and plugins vulnerabilities. The damage may include data leakage leading to financial loss. Traditional antivirus and intrusion detection systems are not efficient against such attacks. Researchers proposed plenty of detection approaches mostly passive blacklisting. However, a few proposed dynamic classification techniques, which suffer from clear shortcomings. In this paper, we propose a novel approach to detect drive-by download infected web pages based on extracted features from their source code. We test 23 different machine learning classifiers using data set of 5435 webpages and based on the detection accuracy we selected the top five to build our detection model. The approach is expected to serve as a base for implementing and developing anti drive-by download programs. We develop a graphical user interface program to allow the end user to examine the URL before visiting the website. The Bagged Trees classifier exhibited the highest accuracy of 90.1% and reported 96.24% true positive and 26.07% false positive rate.

KEYWORDS

Browser Exploits, Drive-by Downloads, Malware Detection, Plugin Exploits, URL Classification, Web Client Exploits

INTRODUCTION

Everyday Internet users are a target by a large number of attackers who are constantly searching for vulnerabilities to perform various attacks with different motivations and intentions (Harley & Bureau, 2008). Narvaez, Endicott-Popovsky, Seifert, Aval and Frincke (2010) considered drive-by download attacks as one of the most important types of these attacks in which the attacker uses legitimate and illegitimate websites to spread malicious code. A file is downloaded to the user machine without trigger by exploiting a web browser vulnerability. The file usually contains a malicious code that runs on the target computer. This malware could be used to steal confidential data, create a backdoor or serve any imaginable malicious intent. Leit and Cova (2011) believe that drive-by downloads are involved in the spread of most of the recent malware infections.

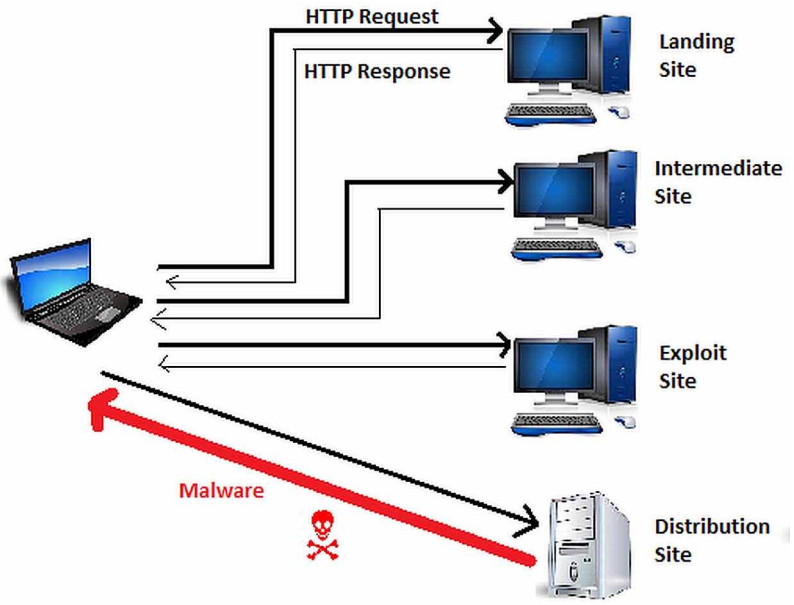
DOI: 10.4018/IJISP.2017100102

Matsunaka, Urakawa, and Kubota (2013) found that the user is simply subjected to this attack by clicking a link in a phishing email, malicious hyperlink, or unwanted popup window. Figure 1 shows one possible scenario to launch a drive-by download attack. First, a malicious website is setup, called the landing or mothership website. This website could be mimicking a legitimate website or actual legitimate website where malicious code is injected. Once the websites are injected with the attack code, they act the first point in a chain of redirections to multiple intermediate websites. The point of the redirections is to hide the actual exploit servers and mislead investigators. The users are finally redirected to the exploit website, which includes a more elaborate malicious code charged with searching for vulnerabilities and flaws based on the version of the user's web browser and operating system. Once vulnerability is located it will be exploited by the malware distribution website to download and install the desired malware directly to user's device without his knowledge. All drive-by attacks need not to follow the exact same flow but the main idea remains valid.

The ultimate goal of drive-by download attack is to take control of the client's system through exploiting the vulnerabilities of web browsers or its extensions forcing it to perform undesirable operations. Takata, Akiyama, Yagi, Hariu, and Goto (2015) found that attacks could result in data or financial loss because the attacker control over the victim's computer. Most drive-by download attacks are carried out by following four main steps. Redirection is considered the first step in which the user is taken through a chain of redirection processes to deliver him to malware distribution site. Attackers use obfuscation as the second step to hide the malicious scripts under several layers of obscurity. The third step is environment preparation in which the attacker seeks getting the permissions to control the memory in order to inject the malicious code in the browser's memory and jumps to execute the injected code. Exploitation is the last step to perform the attack and compromise the vulnerabilities in browser plugins. The compromised client then responds to remote commands from a Command and Control (C&C) run by some bot herder (Cova, Kruegel & Vigna, 2010).

Monitoring traffic and system activities provide a convenient way to identify attacks. Intrusion detection systems (IDS) are used for this purpose. Aldwairi (2006) used Hardware-based IDS to provide efficient and fast detection systems, however they are expensive, complex and are not easy

Figure 1. Drive-by downloads attack flow



to reconfigure. Aldwairi, and Alansari (2011) used software-based IDS with execlusion mechanisim to skip benign packsts. Despite the considerable speedup, it was not as fast as the hardware-based solutions. Researchers considered both the anomaly and the signature-based IDSs. Anomaly-based IDS systems provide a strong protection as they are able to identify new and previously unknown attacks. Aldwairi, Khamayseh, and Al-Masri (2015) found anomaly-based IDSs suffer from high false positives and negatives. Signature-based IDS systems provide higher accuracy in identifying attacks, however they are limited to the predefined patterns and they cannot identify zero-day attacks that are previously unknown.

Many techniques have been proposed and developed by researchers to detect and prevent drive-by-downloads attacks. At the same time, attackers are improving and developing their techniques and approaches to evade detection by existing systems. One approach is to use traditional anti-virus tools that use the stored patterns of common malicious scripts to identify the malicious webpages. Another approach is low interaction honeyclients, which simulate a regular browser behavior. Both approaches are limited to their database, which may not include all new signatures and behaviors. High interaction honeyclients considered to be more advanced than the previous approaches; it relies on monitoring the system environment on a virtual machine. Unfortunately, this approach cannot function until the user install and activate the vulnerable component (Aldwairi & Ekailan, 2011). The most popular approach is using blacklists where all malicious URLs are stored in a database. Before the user visits a webpage, his browser enquires whether this webpage is listed in the blacklist or not, and accordingly the most appropriate action is taken (Priya, Sandhya & Thomas, 2013). However, malicious domains pop in and out very quickly, which makes blacklisting infeasible solution.

In this paper, we propose a novel approach to detect drive-by download infected webpages based on the extracted features from their source code. A data set of 5435 webpages is used for training and testing of 23 different machine learning classifiers and based on the detection accuracy we selected the top five to build our detection model. The approach is expected to serve as a base for implementing and developing anti drive-by download browser extensions.

The rest of this paper is organized as follows. The following section describes the studies related to drive-by download attacks detection. An explanation of the selected features from the source codes and how the identification is done is presented next. The process of collecting the dataset that is used for the training and testing of machine learning classifiers and the research approach follow. The subsequent section presents the findings, a thorough analysis and comparison to related work. Finally, we present our conclusions and suggestions for future directions.

LITERATURE REVIEW

The following lists the most notable and recent proposed techniques by researchers in the area of characterization, detection and analysis of drive-by download attacks. We briefly discuss each approach and point out its advantages and disadvantages.

Cova et al. (2010) proposed a novel approach for the analysis and detection of malicious JavaScripts. The proposed approach was implemented in a tool called JSAND to identify malicious JavaScript code by combining anomaly detection with HtmlUnit emulation. The features identified by the system were based mainly on variable definitions and initializations. The researchers proved the efficiency of JSAND by testing it on a great number of webpages. However, they failed to examine the recent and common techniques used by hackers such as zero pixel objects as well as common functions known to be used for loading malicious content to victims' computers.

Priya et al. (2013) proposed a static approach with a parser written in MATLAB based on the elicitation from the generated databases of the web contents. Few classifiers were used to predict the decision for less than 1000 manually collected URLs. They used the following classifiers: K-Nearest Neighbor, Regression Tree, and Support Vector Machine. The selected features are rather simplistic such as number of tags, number of white spaces and location of elements within the document. In

testing phase, they achieved a success rate of 87.22% with SVM, 87.05% with Regression Tree, and 87.74% with KNN classifier. Small number of features and classifiers, and failing to keep pace with changes in malware attacks is a problem of this approach.

Aldwairi and Alsalman (2011, 2012) introduced a system that identifies malicious websites based on URL lexical and host characteristics such as: length of the top level domain (TLD), number of dots in URL, JS-Enable-Disable, Term Frequency and Inverse Document Frequency (TF-IDF). Naïve Bayes classifier was used to distinguish malicious from benign websites. A successful implementation was achieved with accuracy above 87% and low overhead by using 31 features that are extracted from the websites and examined by the machine learning system. Calculating some of the features is rather expensive and the results can be more accurate with a larger set of URLs.

Matsunaka, Urakawa and Kubota (2013) proposed an approach to prevent drive-by downloads by observing the web transition behaviors. They implemented a framework consisting of two components: monitoring sensors and analysis center. Monitoring sensors are deployed on the client end such as web browser, proxies, and DNS servers while the analysis center is deployed at the network end. They derived an algorithm that detects the download of malwares by utilizing monitoring sensors. Based on the number of transferred hosts, the algorithm applies two conditions, which are: whether the HTTP header or embedded JavaScript indicate the redirection. If neither condition is valid then the algorithm triggers an alert that the file or link is malicious. This approach is based only on webpage transitions and redirections, which gives a misleading classification when advertisements are present on the webpages (Matsunaka, Kubota & Kasama (2014).

Le, Welch, Gao and Komisarczuk (2013) presented a system to detect drive-by download attacks and to analyze the malware behavior in a real-time environment. Even though the system is based on a real environment but the attacks used in the experiments are generated using Metasploit. A real dataset of attacks would better validate the findings. Moreover, the proposed detection mechanism generated an overhead time of 9 seconds, which is considered relatively high, compared to other available systems.

Takata, Akiyama, Yagi, Hariu, and Goto (2015) developed a system aimed to extract the destination URLs from webpages independently of the operating system or the browser used by the user. Based on multiple stages the system extracts the URLs to identify and analyze them. Unfortunately, the system does not perform efficiently if the extracted slices had many variables and functions. That is considered a great disadvantage for this approach because in drive-by downloads attackers attempt to hide their malicious code within the webpage by creating many variables and functions.

The system developed by Kikuchi, Matsumoto and Ishii (2015) classifies drive-by downloads based on features such as redirection methods and the object size. They visualize the connections between the server and the victim where the malware attack is plotted as a red line. They implemented decision trees learning for detecting the malicious connection. This approach is working under the principle that drive-by downloads infected pages will take longer time to load than benign webpages. However, they did not account for benign webpages that contain for example flash objects or high-quality images. Those will take a long time to load, therefore the proposed approach will incorrectly classify them as malicious.

A browser extension designed by Kishore, Mallesh, Jyostna, Eswari, and Sarma (2014) to detect drive-by download attacks by monitoring the webpages before loading by the users. If the page is recognized by the system as malicious it will be reported to the user and according to his decision an action will be taken. This approach depends only on predefined malicious codes. Consequently, it is considered a signature-based detection system, which is effective in only detecting well-known attacks. Unfortunately, attackers keep updating their techniques continuously, therefore signature based detection might not be the most favored approach.

Le, Welch, Gao and Komisarczuk (2013) proposed a system that keeps tracking Internet Explorer 6 to detect the download popups by using window, mouse, and keyboard tracking modules. After the detection and tracking process, the system takes action based on the file filter module. The main

shortcoming of this system is that it does not consider downloads that occur in the background without the user's, knowledge, which is the case in most drive-by download attack scenarios. Moreover, this system is based on Internet explorer 6, which is obsolete.

FEATURES SELECTION

HTML tags are the basic elements for formatting and displaying webpages, these tags play an essential role in distinguishing between malicious and benign sites. Based on HTML tags a number of features can be identified in order to separate trusted webpages from other malicious pages (Ozono, Shiramatsu & Shintani, 2010).

We gathered over 5000 websites using Google.com search engine and verified them through (Alexa, 2016). Before we can test our approach using this dataset we are faced with a dilemma. The feature identification dilemma is how to calculate the features without visiting the website. Visiting a website and opening an infected webpage in the browser means running the embedded code and unleashing a series of events leading to the installation of malware on the computer. We propose an alternative method to open the pages without running the embedded code. We wrote a MATLAB parser program to download and read HTML codes from the URLs in the dataset. Using this MATLAB parser as a features identifier we were able to safely extract numerous features that are very useful and efficient in distinguishing malicious from benign sites.

Initially we experimented with 26 features, however, having more features does not always result in better accuracy. Eleven out of the initial 26 were eliminated because they resulted in misleading training model, which affected the test results. The parser extracts a total number of 15 features and stores them in Excel spreadsheet to be used for further detection and analysis by machine learning classifiers. The extracted features include: white spaces on a page, number of tags on a page, number of links in a page, number of variables definitions, number of loops, number of iframes, number of eval() which interprets a string as code, number of setTimeout, number of HTML redirections, number of opening tag, number of function calls, number of forms, number of input fields, zero pixel objects and on-load() functions.

We introduce new features not previously used by any researchers such as: on-load() functions, zero width i-frame and zero pixel objects. Many attackers recently used those functions or objects to run their malicious code once the page is loaded. The results proved this feature set to be very positive in identifying drive-by downloads. In addition, attackers recently started assigning only zero width iframe or only zero height iframe. Finally, many recent drive-by attacks embedded iframes inside tables with zero dimensions or visibility equal hidden. These new techniques are not detectable by the traditional features identification systems, unlike our approach using on-load() functions and zero pixel objects detection system, which considers extra possibilities. Algorithm 1 gives the step-by-step pseudo code for features extraction program.

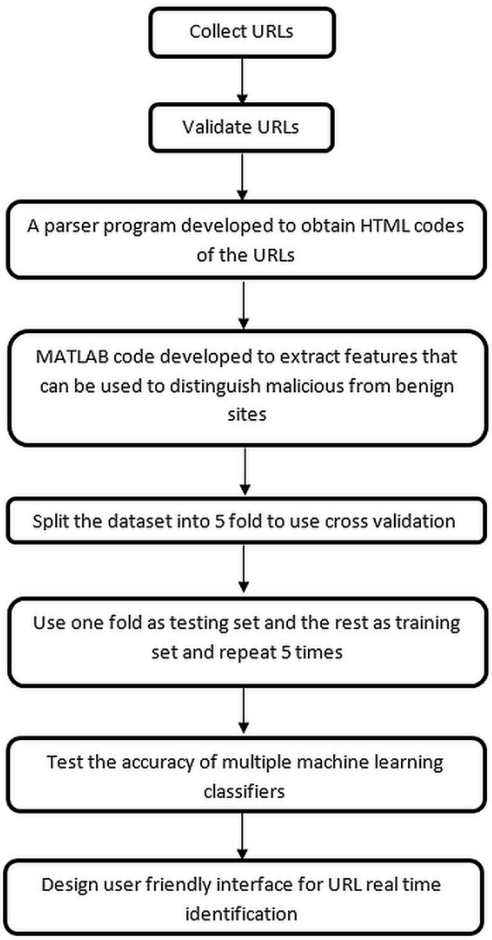
PROPOSED APPROACH

This section provides a description of the proposed approach, dataset collection process, classifiers used for the analysis and the development of the final GUI program. Figure 2 shows the flow diagram with the proposed system various stages. The first step is the collection of the URLs dataset that are classified to be benign or malicious according to (Alexa, 2016). Then those URLs are validated to make sure they still exist and will not return default server error page. Next, MATLAB is used to obtain the HTML source code of these URLs. Then the features extractor code parses the returned HTML code to extract the features and generate a spreadsheet file that will be used as an input to the training system. For validation, the dataset was divided into 5 folds. We examine the dataset with 23 different classifiers, the results of the testing are sorted using accuracy and the best 5 are chosen. Finally, a GUI program was developed to let the user inspect URLs based on the top 5 classifiers.

Algorithm 1. Feature extractor

```
1:   procedure Feature Extract
2:   Input the number of parsed URLS
3:   Initialize Total Array to zeros
4:   Initialize URLs number counter to one
5:   While URLs number counter less than the number of parsed URLs
6:       Load the TXT file with the name of URLs number counter
7:       Save TXT file content to STRING
8:       Calculate the features
9:       Append all the calculated values to Total Array
10:      Create Excel file and save the array to it
11:  end While
12:  end procedure
```

Figure 2. Proposed system flow diagram



Data Collection

Classification systems that are based on machine learning are dependent on the quality and the number of instances provided by the dataset (Freitag, 2000). Obtaining a good quality dataset was a challenging task, because most of the other researchers collected their own URLs. We start by collecting two sets of URLs that are preclassified into malicious or benign. The dataset is saved into an Excel spreadsheet file. Then the MATLAB parser code sequentially imports URLs from the Excel file, downloads their HTML source code and save them in TEXT files numbered from 1 to 5435. The URLs are then validated to make sure they return actual website and no default pages. This code provided a secure, fast and convenient way for collecting the dataset. The pseudo code for the parser program is shown by Algorithm 2.

The URLs for the benign webpages include 1181 links that are gathered from (Alexa, 2016) and Google search engine. On the other hand, the 4254 URLs for the malicious webpages are collected from (hpHosts, 2016), (Quttera, 2016), and Malware Domain List (Malwaredomainlist, 2016). All of the 5435 links were tested successfully and confirmed to be valid.

The last step in building the dataset is to extract the features from the downloaded URLs and store the result of the analysis in one Excel spreadsheet file. Each row in the excel sheets represents a website content and each column a value for a feature.

Classification

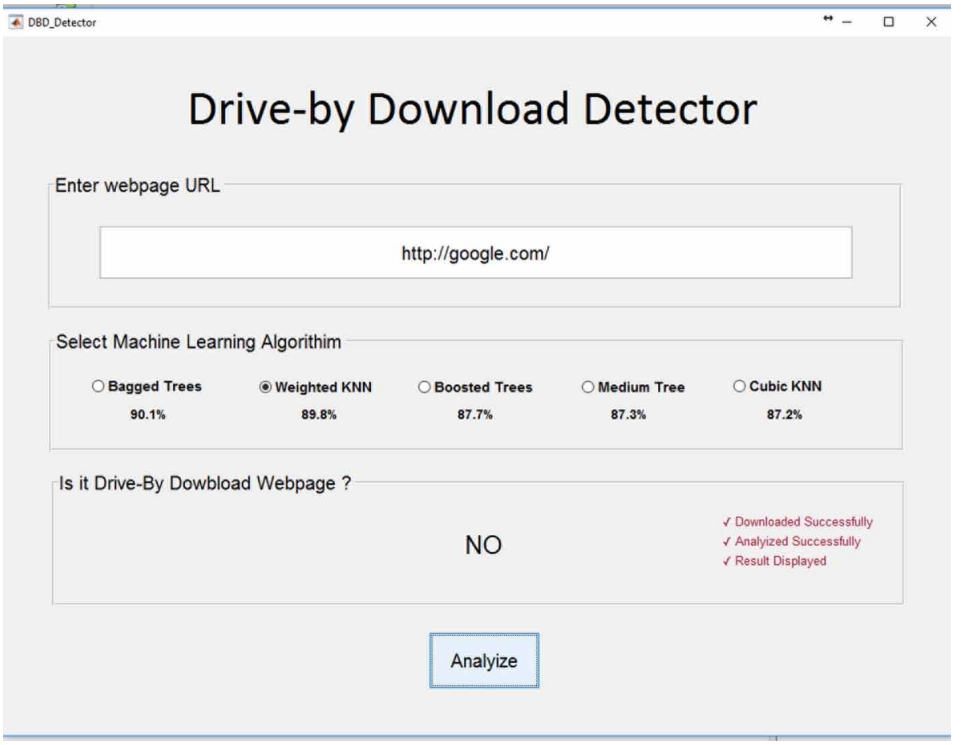
We use MATLAB classification learner in order to explore different classifiers while considering cross-validation of the dataset for the training and testing. Several classifiers were used and those with highest accuracy are selected for use in malicious sites detection. We examine 23 different classifiers and based on the detection accuracy we selected the top 5 to be used for building the final detection model. The 23 examined classifiers are: Simple Tree, Medium Tree, Complex Tree, Linear Discriminant, Quadratic Discriminant, Logistic Regression, Linear SVM, Cubic SVM, Fine Gaussian SVM, Medium Gaussian SVM, Coarse Gaussian SVM, Fine KNN, Medium KNN, Coarse KNN, Cosine KNN, Cubic KNN, Weighted KNN, Boosted Trees, Bagged Trees, Subspace Discriminant, Subspace KNN, and RUSBoosted Trees.

Based on the results of the simulations, a confusion matrix was calculated and the top 5 classifiers in terms of detection accuracy are chosen. We develop a GUI detection model shown by Figure 3,

Algorithm 2. Parser

```
1:  procedure Parser
2:  Read URLs Excel spreadsheet file
3:  Calculate the number of URLs
4:  Initialize URLs counter to one
5:  While URLs counter less than URLs number
6:      Download URL HTML content
7:      Save HTML content into STRING
8:      Create TXT file with name equal to URLs counter
9:      Save STRING content to the TXT file
10:     Add one to student counter
11: end While
12: end procedure
```

Figure 3. Drive-by download GUI program screenshot



using those classifiers. The top 5 classifiers are: Bagged Trees, Weighted KNN with k=10, Boosted Trees, Medium Tree, and Cubic KNN with k=10. Below we provide more details about each classifier:

1. **Bagged Trees:** An ensemble of complex decision trees that is based on Random Forest, Reduced Error Pruning Tree (REPTree), and J48. Considered to be very accurate, but it requires huge memory usage for large datasets (Shah, Aziz, Arif, & Nadeem, 2015);
2. **Weighted KNN:** A K nearest neighbor classifier that calculates distance weighted average for each parameter from the features vector. The weights are assigned according to the feature performance during the classification (Feraru, & Zbancioc, 2013);
3. **Cubic KNN:** A nearest neighbor classifier that uses the cubic distance metric (Kok, Koronackide Mantaras, Matwin & Mladenic, 2007);
4. **Boosted Trees:** A highly predictive performance model that combines the strengths of regression trees and boosting. Its uses relatively little time and memory (Elith, Leathwick, & Hastie, 2008);
5. **Medium Tree:** A medium complexity decision tree with fewer leaves (Quinlan, 1986).

Figure 3 shows the graphical user interface that allows the end user to examine the link before visiting the website. It is envisioned that this GUI would eventually be converted into a plugin integrated into the browser allowing the user to decide whether to visit a website or not without being affected by the malicious code. The program prompts the user to select the classifier from the top five and to enter the suspected link for testing. Next, it will extract and calculate the features discussed earlier. Then test the given link based on the features of its HTML source code using the pre-trained classifier. The program will display to the user the result of the analysis to inform him if the link is benign or malicious. Algorithm 3 shows the pseudo code for the GUI based classifier. Based on the

Algorithm 3. Drive-by download GUI

1:	**procedure** DbD GUI
2:	**Select** the classifier
3:	**Load** the classifier data structure
4:	**Input** the URL
5:	**Download** URL HTML content
6:	**Save** HTML content into STRING
7:	**Calculate** the features
8:	**Create** Total array and assign all calculated numbers to it
9:	**Call** predict function of the structure while sending total array
10:	Display the classification/prediction result
11:	**end While**
12:	**end procedure**

selected classifier the program will load the trained data structure and use it to determine response. The HTML source code of the URL in question will be downloaded and saved into string. That is, it will not be executed or interpreted. Then, 15 features will be calculated on the string with HTML code and the results will be assigned. The array is passed to classifier that returns the final verdict.

EXPERIMENTS AND EVALUATION

The results obtained from classification learner in MATLAB for the 23 different classifiers showed promising accuracy results. We present the results for the top five classifiers with the highest accuracies. Figure 4 shows the confusion matrix, true positive, false positive and accuracy for Bagged Trees, Weighted KNN, Boosted Trees, Medium Tree, and Cubic KNN.

The confusion matrix contains the details about the predicted and actual classification done by the classifier. As shown in Figure 5, the elements of this matrix include True Positive, False Negative, False Positive, and True Negative. The higher the values in the main diagonal reflect better accuracy in the classification. That is higher true positive is the percentage of the correctly predicted YES of the results. And higher true negative is the percentage of the correctly predicted NO of the results.

The Bagged Trees classifier resulted in the highest accuracy percentage of 90.1%, while reporting true positive rate of 96% and false positive rate of 33%. That is 541 (FP and FN) URLs were incorrectly

Figure 4. Confusion matrix, TP, FP and accuracies for top 5 classifiers

Classifier	Confusion Matrix	TP Rate (%)	FP Rate (%)	Accuracy (%)
Bagged Trees	796 385 156 4125	96	33	90.1
Weighted KNN, k=10	756 425 131 4150	97	36	89.8
Boosted Trees	654 527 147 4134	97	45	87.7
Medium Tree	660 521 174 4107	96	44	87.3
Cubic KNN	664 517 184 4097	96	44	87.2

Figure 5. Guide to read confusion matrix

		Prediction outcome P	Prediction outcome N	Total
Actual value	P'	True Positive	False Negative	P'
	N'	False Positive	True Negative	N'
	Total	P	N	

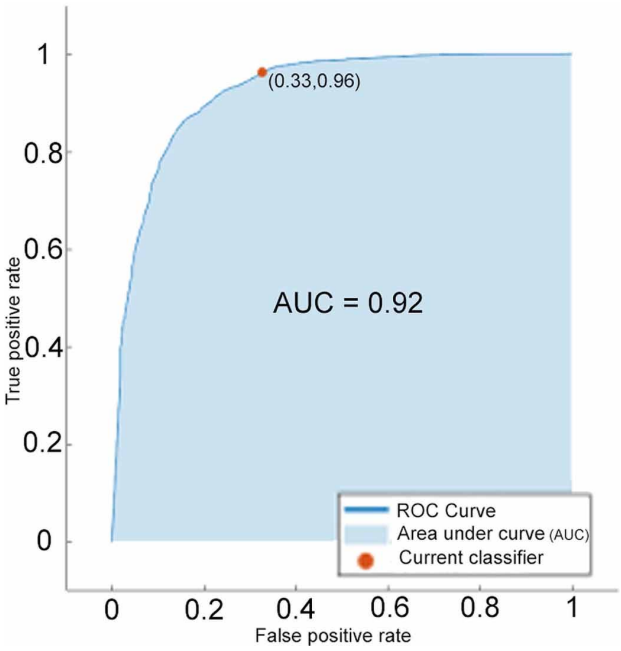
classified while 4921 (TP and TN) URLs were correctly predicted. Same analogy can be applied to read Figure 4 for the other four classifiers.

Figure 6 shows the receiver operating characteristic (ROC) curve for bagged trees classifier. The ROC curve for bagged trees is very close to 90º angle with very sizable area under the curve. The area under the ROC curve (AUC) is 92% which represents the probability the classifier will score a benign website higher than a malicious one.

CONCLUSION

This research dealt with extracting HTML features and analyzing them to detect drive-by download attacks among visited webpages. The idea was to provide a safe utility or plugin that will allow users to test URLs without being affected by the malicious code. Fifteen features where identified of which the new ones are proposed. The HTML source code of 5435 webpages was extracted and

Figure 6. Receiver operating characteristic curve for Bagged Trees classifier



their features were calculated to develop training and testing dataset. A parser and features identifier extracts programs were written and presented.

The performance of 23 different machine learning classifiers was evaluated on the collected dataset using MATLAB. Based on the results top five classifiers further examined and the results presented. Bagged Trees classifier was achieved the highest accuracy with overall detection rate of 90.1%. The area under the ROC curve for Bagged Trees was 92%

Graphical user interface program was developed to allow the end user to examine the link before visiting the website. To enhance the performance of the system, more dataset instances will be acquired to increase the accuracy of the program on its training phase. A two-level security browser extension is currently being developed using our proposed approach to provide real-time user protection.

ACKNOWLEDGMENT

This work was supported by Zayed University Research Office, Research Cluster Award # R17079.

REFERENCES

Aldwairi, M. (2006). *Hardware-efficient pattern matching algorithm and architectures for fast intrusion detection. Available from NCSU Theses and Dissertations Institutional Repository (id 1840.16/3558)*. Raleigh, NC: North Carolina State University.

Aldwairi, M., & Alansari, D. (2011). Exscind: Fast pattern matching for intrusion detection using exclusion and inclusion filters. *Proceedings of the 2011 7th International Conference on Next Generation Web Services Practices (NWeSP),* Salamanca, Spain (pp. 24-30). doi:10.1109/NWeSP.2011.6088148

Aldwairi, M., & Alsalman, R. (2011). MALURLs: Malicious URLs Classification System. *Proceedings of the International Conference on Information Theory and Applications* (pp. 120).

Aldwairi, M., & Alsalman, R. (2012). Malurls: A lightweight malicious website classification based on url features. *Journal of Emerging Technologies in Web Intelligence*, *4*(2), 128–133. doi:10.4304/jetwi.4.2.128-133

Aldwairi, M., & Ekailan, N. (2011). Hybrid Pattern Matching Algorithm for Intrusion Detection Systems. *Journal of Information Assurance and Security*, *6*(6), 512–521.

Aldwairi, M., Khamayseh, Y., & Al-Masri, M. (2015). Application of artificial bee colony for intrusion detection systems. *Security and Communication Networks*, *8*(16), 2730–2740. doi:10.1002/sec.588

Alexa. (2016). Alexa-actionable analytics for the Web. Retrieved from http://www.alexa.com

Cova, M., Kruegel, C., & Vigna, G. (2010). Detection and analysis of drive-by-download attacks and malicious JavaScript code. *Proceedings of the 19th International Conference on World Wide Web* (pp. 281-290). ACM, New York, NY, USA. doi:10.1145/1772690.1772720

Elith, J., Leathwick, J. R., & Hastie, T. (2008). A working guide to boosted regression trees. *Journal of Animal Ecology*, *77*(4), 802–813. doi:10.1111/j.1365-2656.2008.01390.x PMID:18397250

Feraru, M., & Zbancioc, M. (2013). Speech emotion recognition for SROL database using weighted KNN algorithm. *Proceedings of the 2013 International Conference on Electronics, Computers and Artificial Intelligence (ECAI)* (pp. 1-4). doi:10.1109/ECAI.2013.6636198

Freitag, D. (2000). Machine learning for information extraction in informal domains. *Machine Learning*, *39*(2), 169–202. doi:10.1023/A:1007601113994

Harley, D., & Bureau, P. M. (2008). Drive-by downloads from the trenches. *Proceedings of the 3rd International Conference on Malicious and Unwanted Software, MALWARE '08,* Fairfax, VI (pp. 98-103). Retrieved from https://hosts-file.net/

Kikuchi, H., Matsumoto, H., & Ishii, H. (2015). Automated detection of drive-by download attack. *Proceedings of the 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)* (pp. 511-515). doi:10.1109/IMIS.2015.71

Kishore, K. R., Mallesh, M., Jyostna, G., Eswari, P. R. L., & Sarma, S. S. (2014). Browser JS guard: Detects and defends against malicious JavaScript injection based drive by download attacks. *Proceedings of the 2014 Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)* (pp. 92-100). doi:10.1109/ICADIWT.2014.6814705

Kok, J. N., Koronacki, J., de Mantaras, R. L., Matwin, S., & Mladenic, D. (2007). Machine learning. *Proceedings of the 18th European Conference on Machine Learning ECML '07*. Berlin, Germany: Springer. doi:10.1007/978-3-540-74958-5

Le, V. L., Welch, I., Gao, X., & Komisarczuk, P. (2013). Detecting heap-spray attacks in drive-by downloads: Giving attackers a hand. *Proceedings of the 2013 IEEE 38th Conference on Local Computer Networks (LCN)* (pp. 300-303).

Le, V. L., Welch, I., Gao, X., & Komisarczuk, P. (2013). Detecting heap-spray attacks in drive-by downloads: Giving attackers a hand. *Proceedings of the 2013 IEEE 38th Conference on Local Computer Networks (LCN)* (pp. 300-303).

Leita, C., & Cova, M. (2011). HARMUR: Storing and analyzing historic data on malicious domains. *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security* (pp. 46-53). ACM, New York, NY, USA. doi:10.1145/1978672.1978678

Malwaredomainlist. (2016). Retrieved from www.malwaredomainlist.com

Matsunaka, T., Kubota, A., & Kasama, T. (2014). An approach to detect drive-by download by observing the web page transition behaviors. *Proceedings of the 2014 Ninth Asia Joint Conference on Information Security (ASIA JCIS)* (pp. 19-25). doi:10.1109/AsiaJCIS.2014.21

Matsunaka, T., Urakawa, J., & Kubota, A. (2013). Detecting and preventing drive-by download attack via participative monitoring of the web. *Proceedings of the 2013 Eighth Asia Joint Conference on Information Security (Asia JCIS),* Tokyo, Japan (pp. 48-55). doi:10.1109/ASIAJCIS.2013.15

Narvaez, J., Endicott-Popovsky, B., Seifert, C., Aval, C., & Frincke, D. A. (2010). Drive-by-downloads. *Proceedings of the 43rd Hawaii International Conference on System Sciences (HICSS),* Honolulu, HI (pp. 1-10).

Ozono, T., Shiramatsu, S., & Shintani, T. (2010). Preventing fake web pages using push delivery-defending against theft crawlers. *Proceedings of the 2010 IEEE/ACIS 9th International Conference on Computer and Information Science (ICIS)* (pp. 639-644). doi:10.1109/ICIS.2010.81

Priya, M., Sandhya, L., & Thomas, C. (2013). A static approach to detect drive-by-download attacks on webpages. *Proceedings of the 2013 International Conference on Control Communication and Computing (ICCC)* (pp. 298-303). doi:10.1109/ICCC.2013.6731668

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. doi:10.1007/BF00116251

Quttera. (2016). Retrieved from https://quttera.com/

Shah, S. A. A., Aziz, W., Arif, M., & Nadeem, M. S. A. (2015). Decision trees based classification of cardiotocograms using bagging approach. *Proceedings of the 2015 13th International Conference on Frontiers of Information Technology (FIT)* (pp. 12-17). doi:10.1109/FIT.2015.14

Takata, Y., Akiyama, M., Yagi, T., Hariu, T., & Goto, S. (2015). MineSpider: Extracting URLs from environment-dependent drive-by download attacks. *Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC),* Taichung, Taiwan (pp. 444-449). doi:10.1109/COMPSAC.2015.76