CAN SUPPORT VECTORS DETECT EXPLOITS?

NEMANJA MAČEK
School of Electrical and Computer Engineering of Applied Studies, Belgrade; Graduate School of Computer Sciences, Megatrend University, Belgrade; SECIT Security Consulting; macek.nemanja@gmail.com

IGOR FRANC
Belgrade Metropolitan University, Faculty of Information Technologies; SECIT Security Consulting; igor.franc@metropolitan.ac.rs

MILAN GNJATOVIĆ
University of Novi Sad, Faculty of Technical Sciences; milangnjatovic@uns.ac.rs

BRANIMIR TRENKIĆ
School of Electrical and Computer Engineering of Applied Studies, Belgrade; btrenkic@viser.edu.rs

MITKO BOGDANOSKI
Military Academy General Mihailo Apostolski, Skoplje, Macedonia; mitko.bogdanoski@ugd.edu.mk

ACA ALEKSIĆ
Belgrade Metropolitan University, Faculty of Information Technologies; aca.aleksic@metropolitan.ac.rs

Abstract: *An exploit is software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability in operating system or other software products to cause unintended or unanticipated behaviour of computer software, hardware, or other electronic devices. Such behaviour includes actions like unauthorized gaining control of a computer system, unauthorized privilege escalation, or a denial-of-service attack. Although anti-malware products and signature-based intrusion detection systems provide reasonable level of security, they will not detect and prevent execution of new exploits or exploits that tend to evolve, as there is no signature in the anti-malware or intrusion detection database. To raise the overall level of security we have introduced one kernel-based machine learning method, named support vector machines, into an intrusion detection system that is capable of detecting exploits without employing signature database. Experimental evaluation of our solution is conducted on the custom dataset generated in isolated environment.*

Keywords: *Exploits, Machine learning, Support Vector Machines*

1. INTRODUCTION

There is no concise definition of malicious software, frequently referred to as malware in the literature. Malware can roughly be defined as any software intentionally designed with the goal to cause damage to a computer, computer network or anything controlled by a computer system, even industrial power-plants. There are various types of malware, such as computer viruses, worms, Trojan horses, logic bombs, ransomware and cyber-weapons. Malware evolved from early infectious programs, which were written as academic experiments or pranks. Although most of those were typically harmless, they have set a solid ground for development of harmful ones. Today, malware is used by black hats and governments, to steal financial or business information, perform industrial espionage, etc. Further, malware is even used to perform attacks on antagonist country ran industrial plants. One of the first attempts of such activities is the infamous Stuxnet, which was designed to target SCADA systems and is believed to be responsible for causing substantial damage to Iran's nuclear program – over 58% of target system in the early days of infection resides in Iran. Due to the complexity of the malware itself, it is believed that it was built jointly by the United States of America and Israeli government institutions, yet neither country has admitted responsibility for the Stuxnet creation ever since.

Malware does the damage after it is implanted or introduced in some way into a target's computer and can take the form of executable code, scripts, active content, and other software [1]. The authors of malware seek out vulnerabilities in operating systems or computer software, such as buffer-overflow vulnerability that can be exploited. Therefore, one may say that malware is based on exploits, i.e. carefully crafted software chunks that exploit aforementioned vulnerabilities. After exploit is executed on the target system, attacker can take control over the victim, raise privileges, or run the payload attached to the malware, e.g. ransomware encryption module.

A very detailed list of remote exploits, Web application exploits, local and privilege escalation exploits, denial of service and proof of concept exploits is available on Offensive Security's Exploit Database Archive [2]. On July 7, 2018, there were 39,630 exploits archived in the database. Each exploit is very well documented (i.e. which

target it exploits, on which platform, who is the author, etc.), and available for free download as source code (in C, Python, Ruby, etc.)

Although this database may help anti-malware software vendors to write signatures that will help the security software detect an exploit, there are two major issues open: (1) will signature based anti-malware software or intrusion detection system detect exploit-based malware that evolves, and (2) will signature based anti-malware software or intrusion detection system detect zero day exploits? The answer to both questions is, unfortunately, negative.

Having that said, simple anti-malware software or intrusion detection system obviously do not provide sufficient level of defense, meaning that there is a need for additional security mechanisms. One way to add another layer of defense is to employ semi-supervised anomaly detection on the host. This process is based on training the learner with normal behavioral patterns. Although applicable in theory, there are several problems with this approach: it is very hard to obtain all records of normal behavior and draw the exact line between normal behavior and anomaly, normal behavior tends to evolve with time, a noise may exist in the data, etc. Having that said, one may conclude that this approach would lead to a large number of false positives, i.e. legitimate activities that are detected as anomalous.

Another approach is to employ supervised machine learning methods, i.e. systems that are trained with both normal and exploitation data. Although this type of security mechanism can be implemented as host-based intrusion detection system, i.e. system that monitors activities like frequency of system calls and critical system infrastructures, one should note that remote exploit works over a network and exploits the security vulnerability without any prior access to the vulnerable system. Having that said, it is necessary to implement this security countermeasure on the network-level, i.e. create a network-based intrusion detection system trained with data containing of labelled network traffic containing both exploits and normal traffic. As a machine learning classification algorithm we have chosen kernel-based method named Support Vector Machines.

2. MACHINE LEARNING IN INTRUSION AND EXPLOITATION DETECTION

Machine learning algorithms independently collect knowledge from the machine readable information, i.e. they learn from data. Such algorithms build a model from training inputs and use it to make decisions, or predictions [3]. Tom Mitchell provided a widely quoted, formal definition of machine learning: “A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience E” [4]. According to this formal definition, an intrusion detection system (IDS), which we employ to detect exploits, learns to classify events (task T); performance measure P of this task is the classification accuracy, and the experience E is the training set. Mitchell states that machine learning is suitable for application in software engineering when it is necessary to extract knowledge from large databases and when a high degree of adaptation to user needs is required. If we take into account the fact that IDS analyses a large set of events and that it is necessary to adapt IDS to the environment it protects, it can be concluded that machine learning is suitable for use in intrusion detection.

There are two types of machine learning algorithms: unsupervised (no "teachers") and supervised (with "teachers"). Unsupervised algorithms learn from unlabelled examples; the objective of unsupervised learning may be to cluster examples together on the basis of their similarity [5]. Unsupervised learning is suitable for finding patterns in the data. Supervised learning algorithms build a model from a training set (given in the form of feature vectors) with class label assigned to each instance. Once trained, supervised algorithms assign class labels to previously unseen examples of the same task, on the basis of the trained model. Class labels assigned to instances in data sets that are used to train supervised learning based IDS indicate legitimate activities or certain types of intrusions.

Machine learning methods used for classification can be divided into [6]: basic methods (artificial neural networks [7], Support Vector Machines [8], decision trees [9, 10], naive Bayes [11]), hybrid methods (for example, a hybrid of decision trees and naive Bayes – a regular univariate decision tree, where leaves contain a naive Bayes classifier built from the examples that fall at that leaf [12]), incremental methods (naive Bayes updatable), hybrid incremental methods (Hoeffding Tree [13]), basic ensembles (Random Forest [14]), hybrid ensembles (stacking) and hybrid incremental ensembles (Ada Hoeffding option tree). There is a large number of studies reported in the literature that investigate the performances of intrusion detection systems with classifiers based on artificial neural networks (multilayer perceptrons and self-organizing maps), Support Vector Machines, decision trees, Random Forest, Bayesian networks, naive Bayes, hidden Markov models, inductive learning, clustering and nearest neighbors. For more details on findings reported in aforementioned literature, reader may consult [15].

One should note that the nature of input data will influence the choice of classifiers. For example, dimensionality of class label will lead to exclusion of linear regression, multiple linear regression and Support Vector Machines (SVMs), while orientation towards creating simple models lead to exclusion of artificial neural network [10].

3. SUPPORT VECTOR MACHINES

Support Vector Machines are linear learning methods that seek out the decision function in the set of functions (hypothesis) that are linear combinations of input values. The data that is not linearly separable in the original input space is cast into high-dimensional feature space where it is linearly separable. The transformation from the input into the feature space increases the expressiveness of linear methods, but also leads to an increased risk of overfitting. The statistical learning theory [16] defines which parameters should be controlled in order to achieve an appropriate level of generalization and reduce the risk of

overfitting. Maximum margin classifier does not allow learning examples to be misclassified and can only be used with a data set that is linearly separable in the feature space. This constraint motivated the development of soft-margin classifier [17], a modified maximum margin idea that allows examples to be mislabelled.

Support vector machines can be defined as "(1) learning algorithm that uses linear methods (2) in kernel induced feature space, (3) statistical learning theory to control generalization error and (4) optimization theory to solve convex quadratic programming problem; solving this problem equals learning with SVM."

4. FEATURE SELECTION

Feature selection (e.g. features that describe the network connections in the training set) affects the classifier's performance. Although each feature contains a certain amount of knowledge that has an impact towards detection, two facts should be taken into account when selecting features that will be used to form a training set: (1) some features contribute significantly to the classification accuracy, while the influence of others might be almost negligible; (2) system based on an excessive number of features will most probably be CPU-demanding and practically useless if the network flow is heavy.

IDS can be adapted to detect specific categories of attacks by re-adjusting feature weights. There are several methods that can be used to determine feature weights. One feature weight calculation method is based on that idea: the weight of feature is calculated according to the accuracy change of the classifier trained with a set from which feature is removed, compared to the classifier that takes all features in consideration. Another feature weights calculation method is based on F-score. Based on statistic characteristics, it is independent of the classifiers. F-score is a simple technique that measures the discrimination between a feature and the label. Feature's F-score is a ratio of discrimination between the positive and negative sets and discrimination within each of the two sets. The larger the F-score is, the more likely this feature is more discriminative.

To design a high sensitivity exploit detection system based on support vectors, one should follow these steps: pre-process the data (convert the features and normalize feature values), determine optimal hyper-parameters of the original model (optimal hyper-parameters provide a classifier that will predict unknown data most accurately), train the classifier, calculate feature weights, scale training and test set with feature weights (thus the relevance of features towards classification will be incorporated into a model that is trained in the final training step), determine optimal hyper-parameters of the new model, train the classifier with the scaled training set and finally test it.

5. GENERATING THE DATASET

The dataset used in this research is built from traffic captured on the simulated network, consisting of three computers. One computer was used as attacker, the other one as a Linux router which also captured the network traffic using PCAP library, while the third was used as a victim, running Windows and Linux operating systems and software that was found exploitable on Offensive Security's Exploit Database Archive [2]. Synthetic dataset consists of normal, healthy traffic recorded during one day period and variety of simulated attacks, generated by compiled exploits from Exploit Database Archive, fired up with variety of open source and commercial software products. Both healthy and malicious traffic have been recorded separately and cleansed from other protocol and service leftovers (partial noisy data removal), thus leaving clean normal and anomalous PCAP files, which reassembles a scenario for supervised anomaly detection. QoSilent Argus software was used to extract features values from PCAPs and create data instances which were labelled and shuffled into a separate training and test sets. Features used in this research do not include source and destination IP addresses. However, they include flags, connection states, protocols, port numbers and lots of statistical data. Once the feature extraction was done, a sneak peek into the generated CSV certain incompleteness of the dataset. Since Support Vector Machines operate with numerical data, all features must be converted to numerical (scaled to range [0,1]) without missing values. This leaves a Support Vector Machine learner to be trained and evaluated with incomplete datasets, i.e. with some features removed.

6. PERFORMANCE METRICS

The efficiency of our exploit detection system is given by detection accuracy and false negative rates. True alarm (True Positive, TP) indicates that the system successfully detected the intrusion. False alarm (False Positive, FP) indicates that the system incorrectly identified legitimate activity, recognizing it as an intrusion. Missed alarm (False Negative, FN) indicates that the system incorrectly identified an intrusion, recognizing it as a legitimate activity. True Negative (TN) indicates that the system successfully identified the legitimate activity.

Detection accuracy depends on all positive and negative results of the system and is defined as follows:

$$a = \frac{TP + TN}{TP + TP + FP + FN} \quad (1)$$

False Negative Rate (FNR) is the ratio of false negatives and the sum of true positives and false negatives:

$$FPR = \frac{FP}{TP + FN} = 1 - sensitivity \quad (2)$$

Low incidence of false negative alarms indicates that a small amount of exploits is incorrectly identified as legitimate activities. System with low FNR can be used in critical areas of computer networks where an exploitation attempt may not pass undetected, as it may produce significant damage, while discrimination of legitimate activities can easily be corrected (for example, in a corporate network).

7. EXPERIMENTAL EVALUATION

Performance of the algorithms is experimentally evaluated using MATLAB R2016a with Statistical and Machine Learning Toolbox, version 10.2. Within this research the

following algorithms have been evaluated: Support Vector Machines on incomplete dataset and adaptive models based on Support Vectors built by empirical and F-score based re-weighting on incomplete dataset. Training datasets consisting of five thousand instances with 41 features (incomplete) and 54 features (complete) were used for five-fold cross validation. Balanced sets with thousand instances were used as test-sets. Results for each classifier were listed in Table 1 (results of five-fold cross validation) and Table 2 (results on the test set).

Table 1: Five-fold cross validation results (accuracy and false negative rates)

Classifier	Accuracy (%)	FNR (%)
SVM	92.81%	3.39%
SVM (empirical)	94.38%	2.65%
SVM (F-score)	94.76%	2.41%

Table 2: Test results (accuracy and false negative rates)

Classifier	Accuracy (%)	FNR (%)
SVM	89.12%	4.27%
SVM (empirical)	93.67%	3.11%
SVM (F-score)	94.61%	3.07%

5. CONCLUSION

This paper evaluated the possibility of Support Vector Machines to discriminate exploitation code in network traffic from the rest of the data. The dataset for network-based exploit detection was created on the basis of normal, healthy traffic, and exploits, downloaded and run from Offensive Security Database of Exploits. Due to irresolvable incompleteness problem, support vectors were trained with incomplete dataset with several features removed. By boosting feature weights we have managed to increase detection accuracy and reduce false negative rates. However, by doing so, we have introduced adversarial learning issue – overemphasized weights.

REFERENCES

[1] T. Nash, “An Undirected Attack Against Critical Infrastructure, Vulnerability & Risk Assessment Program” (VRAP), Lawrence Livermore National Laboratory, retrieved July 7, 2018.

[2] Offensive Security’s Exploit Database Archive, https://www.exploit-db.com, last time visited October 15, 2018.

[3] M. A. Hall, L. A. Smith, “Practical feature subset selection for machine learning”, In C. McDonald (Ed.), Computer Science ’98 Proceedings of the 21st Australasian Computer Science Conference ACSC’98, Perth, 4-6 February, pp. 181-191, 1998, Berlin: Springer.

[4]. T. Mitchell, “Machine Learning”, McGraw-Hill Science/Engineering/Math, page 2, 1997.

[5] Z. Ghahramani, “Unsupervised Learning”, In Bousquet, O. et al. (Eds.), Machine Learning 2003, LNAI 3176, Springer-Verlag Berlin Heidelberg.

[6] V. Miškovic, M. Milosavljević, S. Adamović, A. Jevremović, “Application of Hybrid Incremental Machine Learning Methods to Anomaly Based Intrusion Detection”, Proceedings of 1st International Conference on Electrical, Electronic and Computing Engineering IcETRAN 2014, Vrnjačka Banja, Serbia, June 2-5, pp. VII2.3.1-6, 2014.

[7] S. Haykin, “Neural Networks: A Comprehensive Foundation”, 2nd edition, Prentice Hall, 1998.

[8] V. Shawe-Taylor, N. Cristianini, “Kernel Methods for Pattern Analysis”, Cambridge University Press, 2004.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, “Classification and Regression Trees”, Wadsworth, Belmont.

[10] B. Predić, G. Dimić, D. Rančić, P. Štrbac, N. Maček, “Improving Final Grade Prediction Accuracy in Blended Learning Environment Using Voting Ensembles”, Computer Applications in Engineering Education, accepted for publication, in press.

[11] V. Cherkassky, F. M. Mulier, “Learning from Data: Concepts, Theory and Methods”, 2nd ed., John Wiley - IEEE Press, 2007.

[12] R. Kohavi, “Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid”, In KDD, pp. 202-207, 1996.

[13] P. Domingos, G. Hulten, “Mining high-speed data streams”, In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 71-80, 2000, ACM.

[14] L. Breiman, “Random Forests”, Machine learning, 45(1), pp. 5-32, 2001.

[15] N. Maček, “One class of adaptive network intrusion detection systems”, Doctoral dissertation, Faculty of informatics and computing, Singidunum University, 2013.

[16] V. Vapnik, “Statistical Learning Theory”, John Wiley & Sons, 1998.

[17] C. Cortes, V. Vapnik, “Support-vector networks”, Machine learning, 20(3), pp. 273-297, 1995.