LITERATURE SURVEY

Malware and Malware Obfuscation Techniques

Stuxnet [1] is one of the most advanced and sophisticated worms ever written. It was designed by the Intelligence agency in the United States and targeted at nuclear refinement plans in Iran. They released a worm through USB to the general public which latches on to any system and gathers administrative privileges for the system using zero-day exploits. This virus uses a total of 4 zero-day exploits. Then it searches for specific Siemens Centrifuge controllers connected to the system. If it does not exist, the worm destroys itself. Once it has control of the controller, it slowly varies the centrifuge speeds over a period of time so that they are all destroyed within a year. The also increase the gas pressure to form rocks within the centrifuges so that they degrade in quality – slowly but steadily. The worm is hailed as one of the most complicated codes ever written in the history of mankind. It floated on the internet for over a year without being detected by any machine or anti-virus.

Obfuscation [2] is by far the most dominant technique along with Signature methods used by malware for evasion and staying out of reach of anti-malware systems. There are various Obfuscation techniques and this paper analyses some of those methods. The simplest obfuscation technique is the dead code technique. It inserts harmless pieces of code at random to various parts of the payload but does not alter the behaviour in any way. The code is placed at clever locations that skip anti-malware scans. Another complementary method is Register Reassignment. Registers support legacy code. By switching from current standard to older generation of registers often, the Malware becomes very hard to find. Subroutine reordering is a very simple but effective Obfuscation technique. It generates n! variants where n is the number of subroutines. It randomly rearranges the subroutine while having minimal changes on the payload or the behaviour of the malware. Instruction Substitution and Code Transposition are two other methods of obfuscation. Instruction Substitution splits a single instruction into multiple instructions while having the same effect. It effectively changes code with equivalent instructions. Code transposition reorders entire sequences of codes rather than just subroutines.

Private stream searching appears to be an entirely effective method for malware to surreptitiously search and exfiltrate email by resisting malware analysis techniques [3]. Malware designed to save and return messages on a specific sensitive topic will be able to do so without revealing the topic of interest upon analysis; all that will be determined is that it scans email in general. Furthermore, as the paper’s implementation demonstrates, there is nothing to prevent these techniques from being used immediately. The example of PIR-based malware illustrates the more general possibility of malware employing public key obfuscation techniques to hide its behaviour.

The traditional security systems like Intrusion Detection System/Intrusion Prevention System and Anti-Virus (AV) software are not able to detect unknown malware as they use signature-based methods. In order to solve this issue, static and dynamic malware analysis is being used along with machine learning algorithms for malware detection and classification. The main problems with these systems are that they have high false positive and false negative rate and the process of building classification model takes time (due to large feature set) which hinders the early detection of malware. Thus, the challenge is to select a relevant set of features, so that, the classification model can be built in less time with high accuracy. The proposal presents a system that addresses both the issues mentioned above. It uses an integration of both static and dynamic analysis features of malware binaries incorporated with machine learning process for detecting Zero-day malware [4]. The proposed model is tested and validated on a real-world corpus of malicious samples. The results show that the static and dynamic features considered together provide high accuracy for distinguishing malware binaries from clean ones and the relevant feature selection process can improve the model building time without compromising the accuracy of malware detection system.

Malware Detection Techniques

Malware sandboxes [5] are one of the most popular methods used by anti-virus testing engineers to find out if the effects of various malware in order to develop and create sophisticated counter measures. These are virtual testing playgrounds that are built with weak protective measures in order to get affected by a virus. However modern viruses have come up with a very new and innovative method against sandboxes in testing. The virus creators’ aim is to make sure the virus or malware is not caught at the testing phase and will go under the radar of the sandbox. Hence, without a countermeasure, it can release fatal payloads onto real systems for adverse effects. The proposed methodology to evade sandboxes are by ensuring differential behaviour in sandboxes. By detecting the environment, it releases payload conditionally. This publication provides a method for sandboxes to catch such malwares with advanced evasion techniques by introducing wear and tear to the malware. Essentially, the sandbox is made as close to a real-life system as possible – specifically an old system. This is the wear and tear being introduced. It can be event logs, recycle bin size, cache entries, network entries, registry, cookie count and so on.

An analysis on malware detection and removal techniques designed for android devices [6] shows worrisome results. Mobile Industry has grown exponentially at a very large rate in the last few years. This period of time is also marked with an increase in malware count designed for android. However, the detection rates stay the same for the most part. The analysis classifies various malware evasion techniques into three categories – Trivial, DSA or static analysis, and Undetectable. The malware is taken through one of these transformations and passed into a system with an anti-malware setup on it. The study came up with many findings. The major takeaway is that all anti-malware systems on android are vulnerable to many transformations and are not up to standard. The second finding is that android anti-malware works primarily based on code level artefacts – package names, asset names etc. However, at least 43% of malware don’t use these techniques at all and hence have an easy path into the system. 90% of the malware designed did not need protection against static analysis of bytecode. The android security system does not bother implementing this vital layer on their systems and hence it is a waste of resource to enable evasion techniques for this method.

In the light of works done in regard to semantics aware methodology[7] for malware detection, aim for such technique is to understand and detect the presence of any malicious intent in a given program. Malware is known to evolve itself through the process of polymorphism or metamorphism in order to evade detection. This procedure is closely referred to as program obfuscation. Adding slightly new behavioural changes is said to modify the malware to a level where it may not be detected. The malware detection systems used a simple pattern matching technique that identified a certain sequence of instructions that is labelled malware using regular expressions. Understanding semantics of the program instruction will help overcome deficiencies brought on by the above mentioned basic technique. This also eliminates the need to frequently update the database used by the commercial virus scanners for all updates of the malware. Here, we learn the context of malicious behaviour is found from the instruction sequence. When a template and an instruction sequence are executed from a state where the contents of the memory are the same, then after both the executions the state of the memory is the same. In other words, the malicious behavior specified by the template is demonstrated by the instruction sequence. Our malware detection algorithm AMD works by finding, for each template node, a matching node in the program. Nodes from the template and the program match if there exists an assignment to variables from the template node expression that unifies it with the program node expression. Once two matching nodes are found, we check whether the def-use relationships true between template nodes also hold true in the corresponding program (two nodes are related by a def-use relationship if one node’s definition of a variable reaches the other node’s use of the same variable). If all the nodes in the template have matching counterparts under these conditions, the algorithm has found a program fragment that satisfies the template and produces a proof of this relationship. Experimental evaluation of this algorithm has shown ability to detect all variants of certain malware, has no false positives, and is resilient to obfuscation transformations generally used by hackers.

Next study involved understanding the working of a malware detection system used to understand and perform behaviour based analysis [8] of malware on android systems called the crowdroid. In this system , a lightweight client is downloaded and used on smartphones as opposed to the security tools and mechanisms used in computers which are not feasible for applying on smartphones due to the excessive resource consumption and battery depletion. Then, the remote server will be in charge of parsing data, and creating a system call vector per each interaction of the users within their applications. Thus, a dataset of behavior data will be created for every application used. The more users using our Crowdroid application, the more complete and accurate will be our system. Finally, we cluster each dataset using a partitional clustering algorithm. This way we can differentiate between benign applications that demonstrate very similar system call patterns, and malicious trojan applications that, even if having the same name and identifier, have a different behavior in terms of distance between example vectors. It is quite often seen that open(), read(), access(), chmod() and chown() are the most used system calls by malware. A benign application could make moderate or heavy use of those system calls and thus trigger false positives. We need to manage the perception of loss of privacy when supporting research community with their behavior information, against the benefit of having access to up-to-date behavioral-based detected malware statistics.

Normalized Compression Distance (NCD)[9] is a tool that uses compression algorithms to cluster and classify data in a wide range of applications. The author has demonstrated that several compression algorithms, lzma, bz2, zlib, and PPMZ, apparently fail to satisfy the properties of a normal compressor, and explored the implications of this on their capabilities for classifying malware with NCD. More generally, they have shown that file size is a factor that hampers the performance of NCD with these compression algorithms. Specifically, they have found that lzma performs best on this classification task when files are large (at least in the range we explored), but that bz2 performs best when files are sufficiently small. They have have also found zlib to generally not be useful for this task. PPMZ, in spite of being the top performer in terms of idempotence, did not come close to the most accurate compressor in any case. Finally, we introduced two simple file combination techniques that improve the performance of NCD on large files with each of these compression algorithms.The author has explored the relationship between some of these properties and file size, demonstrated that this theoretical problem is actually a practical problem for classifying malware with large file sizes, and proposed some variants of NCD that mitigate this problem.

Malware detection systems have been driven by models learned from input features collected from real or simulated environments[10]. A potential malware sample, suspicious email is deemed malicious or non malicious based on its similarity to the learned model at runtime. However, the training of the models has been historically limited to only those features available at runtime but in this paper the author has considered l an alternate learning approach that trains models using “privileged” information–features available at training time but not at runtime– to improve the accuracy and resilience of detection systems. In particular they have adapted and extended recent advances in knowledge transfer and lpmodel influence to enable the use of forensic or other data unavailable at runtime in a range of security domains. The evaluation shows that privileged information increases precision and recall over a system with no privileged information: we observe up to 7.7% relative decrease in detection error for fast-flux bot detection, 8.6% for malware traffic detection, 7.3% for malware classification.

The author introduce ITect[11], a scalable approach to malware similarity detection based on information theory. ITect targets file entropy patterns in different ways to achieve 100% precision with 90% accuracy but it could target 100% recall instead. It outperforms VirusTotal for precision and accuracy on combined Kaggle and VirusShare malware. ITect opens a new front in the arms race. Its level of abstraction makes it difficult to counter and it offers scalability advantages. The authors have demonstrated excellent precision and accuracy on a representative mixture of malware types drawn from the Kaggle malware data and VirusShare. They have demonstrated that both of its constituent detectors, EnTS and SLaMM, outperform previous information theoretic similarity measures. Indeed, ITect outperforms existing AntiVirus engines (as represented in VirusTotal) for accuracy and precision. Its time complexity is bounded above by the number of files to be classified. As an automated, execution agnostic, string-based similarity metric it offers wider scalability advantages beyond its time complexity class alone – reducing human effort and reducing the need for dynamic or static analysis.

The use of machine learning and deep learning in malware detection systems

With the advent of cheap computing power available freely for the general public, malware detection has become near impossible [12]. Malware uses various techniques such as Encryption, Oligomorphic, Polymorphism, Metamorphism, Obfuscation, Fragmentation, Session splicing, Protocol Violations, Code reuse Attacks and more. The most common malware detection techniques used are Signature method, Behaviour method, and Heuristic method. However, these are simply not enough to catch the complex multi layered evasion techniques that most malwares use nowadays. The method being used here is a neural network to catch trojans. The suspected payloads are injected as weights into the neural network. The network catches the trojan when the network receives a very specific type of data.

This next research work [13] gave us an insight into a couple of data mining and machine learning models used to understand malware and detect them prior to affecting the system. Although classification methods based on shallow learning architectures, such as Support Vector Machine (SVM), Na¨ıve Bayes (NB), Decision Tree (DT), and Artificial Neural Network (ANN), can be used to solve the above malware detection problem, deep learning has been demonstrated to be one of the most promising architectures for its superior layerwise feature learning models and can thus achieve comparable or better performance. Typical deep learning models include stacked AutoEncoders (SAEs), Deep Belief Networks with Restricted Boltzmann Machine, Convolutional Neural Networks etc. In this paper, we explore a deep learning architecture with SAEs model for malware detection. The SAE model is a stack of AutoEncoders, which are used as building blocks to create a deep network. An AutoEncoder, also called AutoAssociator, is an artificial neural network used for learning efficient codings. Architecturally, the form of an AutoEncoder is with an input layer, an output layer and one or more hidden layers connecting them. The goal of an AutoEncoder is to encode a representation of the input layer into the hidden layer, which is then decoded into the output layer, yielding the same (or as close as possible) value as the input layer.To form a deep network, an SAE model is created by daisy chaining AutoEncoders together, known as stacking. To use the SAEs for malware detection, a classifier needs to be added on the top layer. In this application, the SAEs and the classifier comprise the entire deep architecture model for malware detection.

The involvement of deep learning architectures such as neural networks in the classification of malware [14] is the insight given by this next research work. Results show that deep learning indeed brings improvements in classification of malware system call traces. Combining convolutional and recurrent layers was the superior idea to achieve this improvement. This combination helps us obtain slightly better results than with simpler architectures with only feedforward or only convolutional layers. Using only LSTM recurrent network also does not achieve accuracy as high as we get with our architecture, which can be explained with the relatively short length of the malware execution traces. Although training time for neural network ranges from three to ten hours, on test time the classification is instantaneous. Therefore the neural network approach is very good in case where there is no common need to retrain the model. Overall, this approach exhibits better performance results when compared to previous malware classification approaches.

Malware detection based on machine learning techniques [15] is often treated as a problem specific to a particular malware family but classifying samples as malware or benign based on a single model would be far more efficient. However, such an approach is extremely challenging as extracting common features from a variety of malware families might result in a model that is too generic to be useful. The author in this paper has used four different machine learning techniques — support vector machines (SVM), 𝜒chi-squared test, 𝑘-NN, and random forests using 𝑛-grams as features to evaluate the effectiveness of generic malware models. To summarize, SVM, 𝜒 2 , 𝑘-NN and random forests all performed well for individual malware families. The accuracy for all of these techniques dropped significantly when the models were more generic. Some of these techniques did better than others in the more generic cases. The random forest specifically was the strongest classifier with an accuracy of 88% for the most generic model.

PowerShell is increasingly used by cybercriminals as part of their attacks’ tool chain, mainly for downloading malicious contents and for lateral movement. Indeed, a recent comprehensive technical report by Symantec dedicated to Power-Shell’s abuse by cybercriminals reported on a sharp increase in the number of malicious PowerShell samples they received and in the number of penetration tools and frameworks that use PowerShell. This highlights the urgent need of developing effective methods for detecting malicious PowerShell commands. In the proposed paper, this challenge is addressed by implementing several novel detectors of malicious Power-Shell commands [16] and evaluating their performance. The proposal implements both “traditional” natural language processing (NLP) based detectors and detectors based on character-level convolutional neural networks (CNNs).

The method proposed [17] is an innovative method for detecting malware which uses combined features (static and dynamic) to classify whether a portable executable file is malicious or benign. The method employs two types of neural networks to fit distinct property of respective work pipelines. The first type of neural network used is recurrent neural network that is trained for extracting behavioral features of PE file, and the second type is convolutional neural network that is applied to classify samples. At the training stage, first the static information of a PE file is extracted and sandbox is used to record system API call sequences as dynamic behaviors. Then extraction of static features based on predefined rules and dynamic features out of the trained RNN model. Next they are combined and use well design algorithm to create images. Lastly, the concurrent classifier is trained and validated using images created in the previous steps labeled with 1(malicious) or 0(benign).

References

[1] Baezner, Marie, and Patrice Robin. Stuxnet. No. 4. ETH Zurich, 2017.

[2] You, Ilsun, and Kangbin Yim. "Malware obfuscation techniques: A brief survey." In 2010 International conference on broadband, wireless computing, communication and applications, pp. 297-300. IEEE, 2010.

[3] Bethencourt, John, Dawn Xiaodong Song and Brent Waters.“Analysis-Resistant Malware”. In: NDSS (2008)

[4] Ekta Gandotra, Divya Bansal, Sanjeev Sofat (2016) “Zero-Day Malware Detection”. In: 2016 Sixth International Symposium on Embedded Computing and System Design (ISED).

[5] Miramirkhani, Najmeh, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. "Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts." In 2017 IEEE Symposium on Security and Privacy (SP), pp. 1009-1024. IEEE, 2017.

[6] Rastogi, Vaibhav, Yan Chen, and Xuxian Jiang. "Catch me if you can: Evaluating android anti-malware against transformation attacks." IEEE Transactions on Information Forensics and Security 9, no. 1 (2014): 99-108.

[7] Christodorescu, Mihai, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. "Semantics-aware malware detection." In 2005 IEEE Symposium on Security and Privacy (S&P'05), pp. 32-46. IEEE, 2005.

[8] Burguera, Iker, Urko Zurutuza, and Simin Nadjm-Tehrani. "Crowdroid: behavior-based malware detection system for android." In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15-26. ACM, 2011.

[9] Borbely, Rebecca Schuller. "On normalized compression distance and large malware." Journal of Computer Virology and Hacking Techniques 12, no. 4 (2016): 235-242.

[10] Burnaev, Evgeny, and Dmitry Smolyakov. "One-class SVM with privileged information and its application to malware detection." In 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), pp. 273-280. IEEE, 2016.

[11] Bhattacharya, Sukriti, Héctor D. Menéndez, Earl Barr, and David Clark. "Itect: Scalable information theoretic similarity for malware detection." arXiv preprint arXiv:1609.02404 (2016).

[12] Jhonattan J Barriga, Sang Guun Yoo. “Malware Detection and Evasion with Machine Learning Techniques: A Survey” International Journal of Applied Research 2017

[13] Hardy, William, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. "DL4MD: A deep learning framework for intelligent malware detection." In Proceedings of the International Conference on Data Mining (DMIN), p. 61. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

[14] Kolosnjaji, Bojan, Apostolis Zarras, George Webster, and Claudia Eckert. "Deep learning for classification of malware system call sequences." In Australasian Joint Conference on Artificial Intelligence, pp. 137-149. Springer, Cham, 2016.

[15] Bagga, Naman. "Measuring the Effectiveness of Generic Malware Models." (2017).

[16] Danny Hendler, Shay Kels and Amir Rubin (2018) “Detecting Malicious PowerShell Commands using Deep Neural Networks”. In: arXiv:1804.04177, Cornell University, 2018.

[17] Zhou H. (2019) Malware Detection with Neural Network Using Combined Features. In: Yun X. et al. (eds) Cyber Security. CNCERT 2018. Communications in Computer and Information Science, vol 970. Springer, Singapore