Empirical Malware Research through Observation of System Behaviour

Stefan Marschalek
Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks,
St. Poelten UAS, Austria
smarschalek@fhstp.ac.at

Manfred Kaiser
Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks,
St. Poelten UAS, Austria
manfred.kaiser@fhstp.ac.at

Robert Luh
Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks,
St. Poelten UAS, Austria
& DMU, Leicester, UK
robert.luh@fhstp.ac.at

Sebastian Schrittwieser
Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks,
St. Poelten UAS, Austria
lbschrittwieser@fhstp.ac.at

ABSTRACT

Behavioural analysis has become an important method of today's malware research. Malicious software is executed inside a controlled environment where its runtime behaviour can be studied. Recently, we proposed the concept of not only observing individual executables but a computer system as a whole. The basic idea is to identify malware by detecting anomalies in the way a system behaves. In this paper we discuss our methodology for empirical malware research and highlight its strengths and limitations. Furthermore, we explain the challenges we faced during our research and describe our lessons learned.

Keywords

security, malware, system behaviour

1. INTRODUCTION

Today, personal computers as well as corporate networks are threatened by a plethora of di erent malware. Most commonly, malware is seen o through an anti-virus scanner which uses prede ned patterns (*signatures*) for matching each le on the system against known threats. Those patterns are obtained by an army of malware analysts scattered through antivirus companies all over the world. Apart from scaling issues (recently, Kaspersky reported more than 300.000 new malware samples per day[1]), the main limitation of signature-based malware detection approaches is that only already known malware can be found. However, malware used for targeted attacks is almost always priorly unknown and thus invisible for signature-based malware detection.

To counter this major limitation of malware detection, behaviour-based analysis of dynamically executed software was proposed in the literature (e.g. [12, 2, 6]). A common concept in this area is to run a suspicious executable inside a controlled environment, for example a sandbox, in order to extract its behavioural pro le [5]. This pro le is then compared to previously collected malicious behavioural pro les in order to evaluate a program's maliciousness [4].

In our previous work [8], we went one step further and observed not only the behaviour of individual executables, but an entire operating system with all its events. To this end, we developed an acquisition service which runs on every endpoint in a network and collects system events such as process starts and terminations, le access, registry writes etc. and forwards them to a central analysis server. In this work, we discuss the lessons learned from this empirical malware research methodology, its strengths and limitations, and give an outlook on future research.

The remainder of this paper is structured as follows: In Section 2 we brie y introduce the methodology of our data acquisition service. Section 3 explains the evaluation of system events on the analysis server. In Section 4 we discuss lessons learned from this empirical research methodology. Finally, Section 5 concludes the paper and describes future work.

2. DATA ACQUISITION

Getting access to correct and realistic data is one of the most challenging tasks in empirical malware research. Our methodology of observing the e ects a malicious program on the entire system has made it necessary to perform the research on real machines. Our research data was collected from a company partner's workstations which were used for typical o ce tasks during a data acquisition process spanning several days.

[1] http://usa.kaspersky.com/about-us/press-center/press-releases/new-daily-malware-count-kaspersky-lab-decreases-15000-2015 (last accessed: Jan. 15th 2016)

WWW'16 Companion, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
http://dx.doi.org/10.1145/2872518.2888609.

2.1 Collected Information

The collection of system information begins with the start-up and ends with the shutdown of the system. It is important that data acquisition begins as early as possible to keep the loss of system information to a minimum. Our collector acquires various system events such as process, network, and registry events.

Most important for analyzing system behaviour is the collection of process events. Each executable has its own characteristic behaviour dictated by its core purpose and all executables together de ne the system behaviour. De ning the normal behaviour of a process is far from trivial. A browser, for example, is designed to access web resources for which it will open sockets but also access local les. Other programs { such as word processors { are mainly used o ine, but can also access online help resources, thus require opening a socket as well.

For collecting the system events required to generate a characteristic process behaviour pro le we developed a custom-built kernel monitoring agent which is running on each protected host. The main advantage over existing tools such as $procmon$[2] is its much higher monitoring granularity and that it directly writes the collected data to a central database.

Modern operating systems are able to execute more than one process at the same time and a process can have more than one thread. Each process has its own process id (PID) and knows the PID of its parent process. Besides the PIDs there are many more attributes available for collection. These are the image name, the user context (username, group, owner) and the time of process start. If a process creates a new thread, accesses the registry or opens a new network socket, those events are also stored in the central database.

All events are then used to build a comprehensive tree of running processes which categorizes activities and helps to identify malicious behaviour. We experienced that building a process tree and linking the events to the appropriate process is not as straightforward as it looks at rst. Using the process ID as key is not possible because those IDs are reassigned to new processes after the termination of the former owner [7].

For the actual process tree building described in the following section we used an approach that considers this persistence issue.

2.2 Building process trees for classification

A process tree is a tree-like structure in which all process events and subprocesses are depicted as nodes. In theory the process tree of a system consists of a single root node, but in reality it is not possible to gather all processes immediately after the beginning of the system boot sequence. This results in tree fragmentation (i.e. several independent process trees). We experienced that on a typical o ce workstation there are about ve subtrees. On special purpose systems such as database servers or web servers the subtree count can exceed this number signi cantly. In Figure 1 a sample process tree is shown. Node A represents the SYSTEM process of the operating system. This process is intended to launch all other processes (e.g. the explorer.exe shell of the Windows operating systems) which are required to interact with the system. The data acquisition tool itself is a normal process like all other processes and is started at the earliest possible time during system boot.

[2]https://technet.microsoft.com/en-us/sysinternals/processmonitor.aspx (last accessed: Jan. 5th 2016)

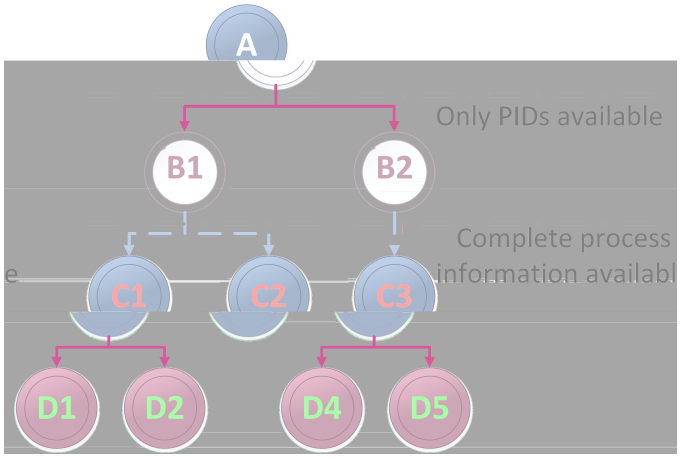

Figure 1: Partial process tree no longer linked through additional attributes

Because of the problem that process IDs are reused by the system [10] we had to develop an algorithm for nding correct parents by checking the process creation time.

Furthermore, building the process tree at runtime can result in an incomplete tree. This can happen when process A starts process B1 which starts process C1. If process B1 was terminated before the tree was built, it is not possible to nd a link between process C1 and process A. The reason is that process C1 knows the process ID of the terminated process B1. However, due to the fact that process B1 was terminated, the parent information is no longer available.

Our methodology can be used to analyze an entire system, beginning with the time the tool was started, or to analyze a single process and all of its children process trees. In cases where only one speci c process is relevant for the analysis, it is not important that the system session is fully recorded. Observing a single process also reduces the amount of data and can still be used for e.g. state comparison.

The building of subtrees is done directly from the database by using Transact SQL and is visualized with dndTree [3, 13, 11].

3. CHOOSING THE RIGHT ALGORITHM

Behavioural pro les can be extracted from sub-process trees using machine learning techniques. At rst we selected meaningful features from the trees and cut everything else away. Then we converted the remaining trees into lists in order to atten the tree-like structure for further use. Since the output of this conversion are lists in the string format we needed an algorithm that can learn from those lists and compare them to each other. We considered a Na•ve Bayes and n-grams approach as both are well-suited for the kind of string-based learning required to analyse our monitoring data. We ultimately decided to use n-grams to retain the chronology of events. An n-gram is a window with the size n, slid over a list with a step size of 1 [1].

3.1 Learning

For comparison of the process lists we chose the Malheur algorithm [9] which is based on n-grams. Malheur is perfor-

mant, can handle large string type reports as well as large amounts of reports and supports incremental analysis. Out of the output of the n-gram algorithm a behavioural pro le is generated for each tree.

During data collection on non-infected systems a baseline of behaviour can be generated. A deviation from this baseline indicates unknown and possibly malicious behaviour.

Throughout our tests we monitored a baseline for a time period of about four and a half days and then tested this baseline against the trees of three distinct malicious traces. Those traces were deemed malicious by the algorithm alongside a total hit ratio of about 99.07 percent for all 7623 trees. That implies a low false positive rate of 0.93 percent or 71 falsely classi ed trees that slightly exceeded the distance threshold. Those rates are comparable to dynamic analysis that has been done before [2].

4. LESSONS LEARNED

In our earlier research [8] we were able to demonstrate that observing an entire system's behaviour is a promising approach for empirical malware research. The lessons learned from our research should help other researchers with implementing similar concepts.

The rst important nding in our research was the enormous importance of working with real endpoint data. Getting access to event data from real workstations is certainly challenging, but essential for research building on that data. In this point our methodology stands in stark contrast to previous dynamic malware analysis approaches which require realistic client activity in a much smaller degree. Collecting this real data was more challenging than initially expected, because of technical limitations such as the impossibility of starting the data collection at an early stage during system boot. As a consequence, it is necessary to cope with a known incompleteness of the data set by observing distinctive subtrees instead of a single system process tree. This approach helps to maintain data integrity and keeps information loss to a minimum. Not all missing data can be reconstructed. Actions performed prior to the start-up of the monitoring tool cannot be accurately recovered and might lead to orphan process trees.

We discovered that only a very small subset of features from the data set is actually required for evaluating the maliciousness of activities while other features can be removed entirely from the set. This fact not only improves completeness of data but also reduces storage requirements. In our experimental setup, we were able to reduce to the total amount of collected endpoint activity data to well below 100 megabytes per day per workstation with an average system load.

5. CONCLUSIONS

The described methodology for empirical malware research works well as a proof of concept and encourages further research. It also shows that the idea of learning benign behaviour rather than malicious behaviour is not only possible but may as well become an important part of malware analysis in the future.

Based on the lessons learned from our earlier research we plan to improve the methodology in the future. Key research questions are the evaluation of automatic feature selection methods, incremental clustering of event data and the adaption of other classi cation approaches such as Na•ve Bayes or graph matching algorithms.

Acknowledgments

The nancial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged.

Our gratitude extends to IKARUS Security Software for their invaluable support.

6. REFERENCES

[1] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. Detection of New Malicious Code Using N-grams Signatures. In *PST*, pages 193{196, 2004.

[2] U. Bayer, E. Kirda, and C. Kruegel. Improving the e ciency of dynamic malware analysis. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1871{1878. ACM, 2010.

[3] R. S. Coutinho. D3.js drag and drop tree. `https://github.com/RodrigoSC/dndTree`.

[4] J. De Vries, H. Hoogstraaten, J. van den Berg, and S. Daskapan. Systems for Detecting Advanced Persistent Threats: A Development Roadmap Using Intelligent Data Analysis. In *Cyber Security, 2012 Intl. Conference on*, pages 54{61. IEEE, 2012.

[5] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys*, 44(2):6, 2012.

[6] A. R. Gregio, D. S. Fernandes Filho, V. M. Afonso, R. D. Santos, M. Jino, and P. L. de Geus. Behavioral analysis of malicious code through network tra c and system call monitoring. In *SPIE Defense, Security, and Sensing*, pages 80590O{80590O. Intl. Society for Optics and Photonics, 2011.

[7] K. Jensen and L. M. Kristensen. *Coloured petri nets: modelling and validation of concurrent systems*. Springer, Dordrecht ; New York, 2009.

[8] S. Marschalek, R. Luh, M. Kaiser, and S. Schrittwieser. Classifying malicious system behavior using event propagation trees. ACM, iiwas2015 conference, 2015.

[9] K. Rieck, P. Trinius, C. Willems, and T. Holz. *Automatic analysis of malware behavior using machine learning*. Journal of Computer Security, 2011.

[10] A. S. Tanenbaum and H. Bos. *Modern operating systems*. Prentice Hall Press, 2014.

[11] M. Wagner, F. Fischer, R. Luh, A. Haberson, A. Rind, D. Keim, W. Aigner, R. Borgo, F. Ganovelli, and I. Viola. A Survey of Visualization Systems for Malware Analysis. In *Eurographics Conference on Visualization (EuroVis) State of The Art Reports*, pages 105{125. EuroGraphics.

[12] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, (2):32{39, 2007.

[13] T. W chner, A. Pretschner, and M. Ochoa. DAVAST: data-centric system level activity visualization. pages 25{32. ACM Press, 2014.