WORKING FORMAL METHODS SYMPOSIUM 5 - 8 July 2017

Automated analysis of possible malware applications

Vlad Craciun
Department of Computer Science
UAIC, Iasi
Email: `vcraciun@info.uaic.ro`

Keywords: *reverse engineering*, *malware analysis*,*static analysis*, *dynamic analysis*, *symbolic execution*, *taint analysis*

1 Abstract

Static and dynamic program analysis are widely used by researchers to identify possible application issues or unwanted execution behaviors. Some researchers are working mostly at source level, making use of static analysis, whereas others are focused more on dynamic analysis and reverse engineering to understand how binary code flows through execution environments. There are already available a lot of intruments to help analysts dig for specific features when dealing with binary applications. JIT translators, decompilers, debuggers, emulators and virtualized environments create together a set of tools for malware researchers. However the number of malwares which security companies are dealing with, in the last few years, have already grown at a point where human intervention is desired only for the fierce ones. A lot of effort was pushed in the last decade, into automated analysis frameworks and tools, circleing around symbolic execution and taint analysis. However only few of them are able to bring light upon the most hidden execution paths. In an attempt to get some informations about execution paths encoded in a simple application, a simple symbolic execution tool would have been enough, but dealing with malware will require special features to face the hiding mechanisms like obfuscation, code-packing, code-encryption, polymorphism and the ability to render useless and exploit through targeted attacks, most of the public available analysis environments. These practices, enables them to know when they are subject to analysis and force the execution to finish.

The challenge with this type of threats is to understand faster what they are up to and the same time to control their anti-analysis tricks.

Most of the automated analysis engines have their roots in a serie of articles published since 2007. A. Moser et al. in [5] uses dynamic symbolic execution (also known as concolic execution) to identify malicious behavior using QEMU. A more complex project, BitBlaze still based on QEMU is developed by D. Song et al. in [6], which uses dedicated analysis plugins to highlight certain features of binary applications and for example, the benefits of symbolic execution were available later by a plugin called Rudder. Another project started around 2010 by V. Chipounov et al. in [14] makes use of concolic execution to selective execute symbolic only the interest paths, thus filtering out symbolic space at system calls. Other kind of frameworks focus more on taint analysis or control flow graph interpretation like [7] to filter out obfuscation and polymorphism. Some others, are using virtualized environments in [8], [9], [10], [11], [13] and [12] to understand the application, outside the execution environment. More recent and improved frameworks are also available. They tend to use pools of execution trace creators like QEMU and PIN and to make use of concolic execution to push the limits of path exploration. Some of them are presented in [1], [2], [3] and [4], however they all can be targetted or exploited as they are pulbic and free to use and study, by malware authors.

The main purpose of the research is to provide a fast way to understand malware actions and if possible, reverse the environmental changes or prevent the execution. We were able up to this point, to develop from scratch, a JIT for binary code translation to help us understand Windows malware faster without loosing additional meaning of the data flow. We achieved this, combining the power of dynamic analysis over a couple of plugins handled by the translator and the power of static analysis to bring back symbols onto the execution traces. In this work, we propose a JIT translator, similar to PIN from Intel along with a couple of plugins which could automate different types of analysis. The advantage of using customized tools and frameworks will help us evade the anti-analysis tricks used by the malware authors, as they are unable to render them useless, unless they get to study them somehow. The types of analyses we can handle, expand also to emulators and virtual environments and cover bacward and forward taint analysis, dynamic symbolic execution, hooking and Control Flow Graphs. We are trying to put all these pieces together to bring binary analysis to the next level of speed and eficiency. Among the entire application, we will focus more on the translator engine and dynamic symbolic execution module, used to explore as much execution paths as possible. The entire analysis engine is still in progress and we have a well working JIT, an almost finished dynamic symbolic execution engine with semantic for the most used arithmetic instructions and some dedicated hooking plugins to control the entire analysis context, thus distributing the analyzer with each new process, thread, callback or exception created by the target binary. We expect to finish soon the caching for environmental changes and full loop translations to optimize the time for trace creation. Caching changes will allow us to isolate what malware expects to find or modify into our environment by what it can actually do and it also speeds up the path exploration for dynamic symbolic execution, as the path shifting will resume to reversing memory and cached environment snapshots. We will also need to see how paths revealed by the dynamic symbolic execution could be labeled with symbols of the executed functions along with their parameters to help us discover cryptolocker behavior and not only, or how operating on control flow graphs may create strategies to clean binaries affected by file-infectors, or detecting certain graphs among merged execution traces.

The presentation will start with a short introduction to static and dynamic analysis benefits and will continue with the JIT design. In the second half we will try to see some differences between static symbolic execution and different ways to do its dynamic version, where we will also try to fit our own approach. In the end there will be highlighted possible extensions of the framework and some of their implications.

References

[1] Shoshitaishvili, Yan and Wang, Ruoyu and Salls, Christopher and Stephens, Nick and Polino, Mario and Dutcher, Andrew and Grosen, John and Feng, Siji and Hauser, Christophe and Kruegel, Christopher and Vigna, Giovanni (State of) The Art of War: Offensive Techniques in Binary Analysis *2016 IEEE Symposium on Security and Privacy* http://angr.io/.

[2] Reverse engineering framework in Python *2015*, https://github.com/cea-sec/miasm.

[3] Jonathan Salwan, Pierrick Brunet, Florent Saudel, Romain Thomas A DBA Framework *2015* https://triton.quarkslab.com/.

[4] A binary tracing tool for Windows *2016* https://github.com/K2/EhTrace.

[5] Exploring Multiple Execution Paths for Malware Analysis *2007* http://analysis.seclab.tuwien.ac.at/papers/explore.pdf.

[6] BitBlaze: A New Approach to Computer Security via Binary Analysis *2008* https://www0.comp.nus.edu/~liangzk/papers/iciss08.pdf.

[7] MAIL: Malware Analysis Intermediate Language: a step towards automating and optimizing malware detection *2013* http://dl.acm.org/citation.cfm?id=2527006.

[8] PyTrigger: A System to Trigger and Extract User-Activated Malware Behavior *2013* http://ieeexplore.ieee.org/abstract/document/6657230/.

[9] MAVMM: Lightweight and Purpose Built VMM for Malware Analysis *2009* http://ieeexplore.ieee.org/abstract/document/5380697/.

[10] AMAL: High-Fidelity, Behavior-based Automated Malware Analysis and Classification *2014* https://link.springer.com/chapter/10.1007/978-3-319-15087-1_9.

[11] Ether: Malware Analysis via Hardware Virtualization Extensions *2008* http://dl.acm.org/citation.cfm?id=1455779.

[12] Toward Automated Dynamic Malware Analysis Using CWSandbox *2007* http://ieeexplore.ieee.org/abstract/document/4140988/.

[13] V2E: Combining Hardware Virtualization and Software Emulation for Transparent and Extensible Malware Analysis *2012* http://dl.acm.org/citation.cfm?id=2151053.

[14] S2E: A Platform for In-Vivo Multi-Path Analysis of Software Systems *2011* dl.acm.org/citation.cfm?id=1950396.