Data Flow analysis in malicious binary codes. Cartography of functionalities embedded in a binary codes and their inter-relations.

Contact: Jean-Yves.Marion@loria.fr

Abstract

The team Carbone at LORIA and thanks to High Security Lab (HSL) has devised a novel method in order to analyse binary codes dubbed Morphological analysis. The morphological method finds code similarities and detects malware. The objectif of this thesis is to reconstruct the data flow graph inside an obfuscated binary code in order to cartography the used functionalities together with the inter-relations between functionalities. The outcome is a contribution to the detection of new threats.

The context of malware detection

Anti-virus companies keep a low profile on how their detection methods work. That said, the standard detection method of malicious codes is to assign a signature to each malware that characterize it. A signature is a regular expression, or quite often a mere sequence of bytes, identifying a malware. Usually, Yara [5] rules are used and a rule looks like {68 3B DB 00 00 ?? ?? ?? ?? 00 ?? FF 15} where ?? corresponds to any bytes. (this sequence is a signature of the ransomware Cerber). All binary files or executed codes containing this signature, that is a sequence of bytes satisfying the regular expression, is then considered as infected. So given a data base of signatures, the detection engine is the part of the anti-virus which searches one of these signatures inside a program. The advantage of this approach is its speed et weak rate of false positive. The drawbacks are mainly that (i) this methods is not able to detect variants or mutation of a known malware, and so a fortiori unable to identify a new threat and (ii) the construction of signature is most of the time done manually.

In order to respond to both difficulties, we have developed a approach that we named Morphological Analysis. Morphological analysis leans on the reconstruction of an abstraction of the control flow graph (CFG). This reconstruction necessitates the combination of both a static and a dynamical analysis. The dynamic analysis consist in executing in a safe environment a program and extracting each wave of codes in order to thwart obfuscation based on code self-modification, which are most of the time produced by packers [2], see also [4]. The static analysis consists in disassembling each code wave and to reconstruct the control flow graph. Then the control flow graph is abstracted and cut in graphlets called sites. The sites compose the behavioural data base that identifies a malicious code. The detection engine of Gorille, which is the named of our tool implementing morphological analysis, searches sites inside the control flow graph of the targeted program.

State of the art of functionality identification

Compare to a model of detection based on signatures or on behaviours, we could propose another approach based in the identification of implanted functionalities in a code and to determine the relationships between those functionalities. The first outcome would provide a valuable help to reverse-engineering. Recall, the sate of the art disassemble IDA just considers the header of a function correctly compiled. The second outcome would to predict malicious behaviours. As an example, take an application that contains a communication function sending encrypted message with base64 used by the APT28.

On x86 obfuscated binary codes, the issue is hard. It consists in recovering semantics of binary codes. There are only a few studies. Let us cite two of them. The first one is to consider the problem has an interpolation [3]. It has been successfully applied to cryptographic primitives. That said, this approach to do not generalize to other classes of functions. The second one [5] uses a symbolic analysis. It is also applied to the identification of cryptographic primitives. That said, the method is today inefficient because of its complexity.

Thesis subject

Morphological analysis could bring a solution. If the site carving is not enough sharp to identify implemented functionalities nowadays, it is still possible to identify some cryptographic primitives and even to detect the compiler and the used options. As a result, the roadmap is quite clear. We should design a method to collect a set of sites that will collectively characterizes a functionality. Today, we have all the tools to extract sites. Consequently, the other "variable" on which we can "play" is the data flow. If this approach was partially explored by [5], we have here two assets: (i) we know how to construct an abstraction that preserves semantics and simplify searches and (ii) we could extend the notion of site on (multi-)graphs of data. In other words, the goal is to recover a data flow graph and to cut it in such a way that, combined with CFG sites, we could be able to identify a functionality. One this issue solved, the next step would be to determine correlations with other identified functionalities. The result would be a cartography of implement functions and their correlations.

References

[1] Bardin, S., R. David, et J-Y Marion. «Backward-Bounded DSE: Targeting Infeasibility Questions on Obfuscated codes.» *IEEE Symposium on Security and Privacy (SP)*, 2017.

[2] Bonfante, G., J. Fernandez, J-Y Marion, B. Rouxel, F. Sabatier, et A. Thierry. «Codisasm: Medium scale concatic disassembly of self- modifying binaries with overlapping instructions.» *CCS*, 2015.

[3] Calvet, J., J. Fernandez, et J-Y Marion. «Aligot : cryptographic function identification in obfuscated binary programs.» *ACM Conference on Computer and Communications Security - CCS* , 2012.

[4] Cheng, B., et al. «Towards Paving the Way for Large-Scale Windows Malware Analysis: Generic Binary Unpacking with Orders-of-Magnitude Performance Boos.» *ACM Conference on Computer and Communications Security (CCS).*

[5] Lestringant, P. «Identification d'algorithmes Cryptographiques dans du code natif.» doctorat Université de Rennes 1.

[6] Yara. *http://virustotal.github.io/yara/.*