EFFECTIVE PREVENTION OF REVERSE ENGINEERING ATTACKS ON SOFTWARE

Jeyalakshmi Jeyabalan, Priya Loganathan and Sree Subha Soundarajan
Department of Information Technology Rajalakshmi Engineering College Chennai, India
E-Mail: jeyalakshmi.j@rajalakshmi.edu.in

ABSTRACT

With increasing availability of knowledge over Internet, it has become so common that any novice can go way beyond his technical ability to crack any licensed software and use it like any legitimate user. Reverse Engineering has been one of the prominent techniques to crack the software with hash code if not the high level language. The paper discusses a novel architecture where the effect of any such hacking effort in nullified. The paper narrates various static and dynamic techniques for anti-reverse engineering and the need for integrating such techniques into the packers or installers. Whatsoever the software and the purpose be, the anti-reverse engineering installers or packaging pieces of code can be equipped with minimal prevention against reversing attacks, which may be mandatory in future . The virtualization techniques are the key to the proposed system. The installer or unpacking software is equipped with the proposed techniques, so that the reverse engineering efforts may be thwarted successfully.

Keywords: anti reverse engineering, software engineering, reverse engineering.

INTRODUCTION

Software is all about evaluating conditions and implementing the cases or paths that follow a particular condition. A successful attacker can make a guess, of those assumptions and acquire an illegal entry into the software. The availability of tools like Hex Editors has been a boon to such attackers. The hex code if understood by the attacker, not having knowledge of the high level language, still enough damage can be done. This reverse engineering effort can be successfully identified and thwarted using few anti reverse engineering techniques. The proposed system suggests a novel architecture for the same purpose. In the proposed system, the installations or unpacking invoke a new virtual machine, in which these activities afore mentioned, happen. The effort proposes integration of anti-reverse engineering techniques to the architecture of software i.e. on installers or packaging software. The activities of the process are closely monitored and if any anomaly detected, the virtual machine is shut down by the hypervisor.

SYSTSEM ARCHITECTURE

The proposed system as described in Figure-1, does the unpacking or installations in a virtual machine and the activities if found suspicious are suspended based on the report generated. The process is checked using static and dynamic techniques for any issues. The policy in the repository is considered, while generating the report. Based on the report the process proceeds or exits.

The existing systems like disassemblers, decompilers, dumpers, etc focus only their scope of activities. But if reverse engineering needs to be successfully detected and prevented, then a wider scope of analysis is required. The proposed system suggests oneness of software and prevention techniques. And it also enforces both static and dynamic techniques for this purpose.

The architecture is briefly explained as follows. The hacker tries to use hex editors and acquires the hex code of the software. The system creates a virtual machine and runs any unpacking and installation only in the virtual machine. The static and dynamic analysis is performed on the hex code. The policy repository stores the policies used to evaluate the process of unpacking. The policies may store subject oriented information like, the time taken to execute any instruction on the software, the coverage that is expected, the independent paths of the software which will be traversed, the use-def patterns for any variable which is legitimate and anomalous, the guard pages, the breakpoints, the ACLs, the debug summary from the debugger etc., The policy based analysis generates a report on the process and if it is found to be deviating from expected results, then the hypervisor simply shuts the virtual machine. Thus it prevents any direct effect on the system. Any such process runs in a sand box, thus quarantining the entire process and its outcomes. If trustworthy, the process continues, else the force shut down process takes place.

The steps and techniques used to monitor the virtual machine are briefed below. The techniques [1] [2] [3] are broadly classified into manual and dynamic techniques.

Static Techniques

These techniques would only rely on hash code walk through in order to find any errors or break ins. These techniques follow the methods mentioned below.

Break Point Analysis

Breakpoints can be manipulated and return addresses can be altered for breaking in. A table containing all breakpoints and their return addresses is maintained namely Breakpoint Analysis Table(BAT) which is compared with the traces collected from the system, can identify breakpoint oriented attacks.

Coverage Analysis

The coverage of the branches, conditions and statements can be maintained for software and any deviation from normal can be checked for issues.

Control / Data flow analysis

The control flow and complexity analysis leads to calculation of independent paths from McCabe Cyclomatic Complexity, which can be used to check the execution of any operation along with the data flow use-def patterns.

Timing Analysis

The timing for any operation is recorded in machine cycles. If any delay is detected, it could be possible break in.

Automatic Technique

The instructions are executed in debug mode, which gives an added advantage of finer control over the executing software. The methods used herein are stated below. In addition to obfuscation the following techniques can be used for prevention.

Guard Pages

Guard pages act like an alarm for memory page access. The guard pages trigger an alarm if any illegitimate access is detected.

Anti-Dumping

Dumping is the process of taking snapshot of the executing traces and trying an attack. Possible protection techniques are encryption, compression, transformations based storage techniques. Nanomites are a famous technique that replaces branches with breakpoints. Information is misplaced and jump can be used to fetch them back. Serial encryption in several levels can make the life lot tougher for a hacker.

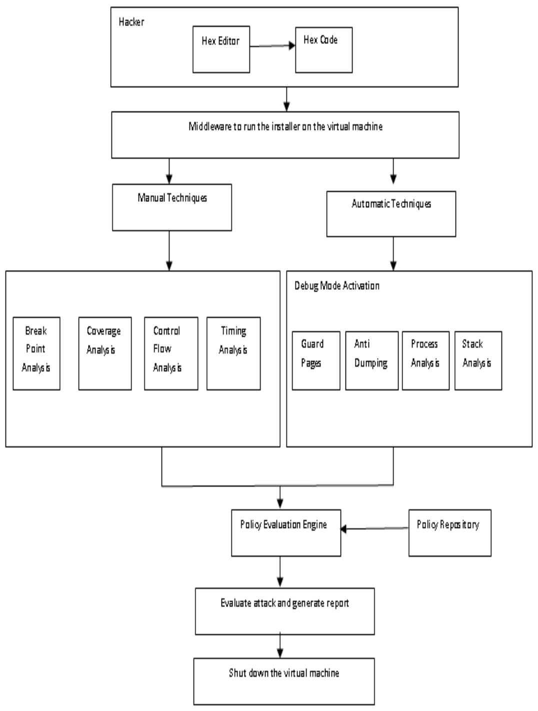

Figure-1. System architecture.

Process Analysis

This intensely relies on process analysis. It can use techniques like open process and self-debugging. These are used to analyze if a debugger is attached to the process, or making the child process debug the parent process.

Stack Analysis

Machine level instructions can be used in order to ensure there is no tampering done. Stack traces can be monitored in cases where stack can be compromised with its own processes.

Access Control Check

The ACL (Access Control List) and user accounts can be scrutinized in order to ensure protection.

Native Code Permutation

The code can be misplaced and records can be maintained in order to confuse the hacker.

The above paragraphs illustrate how the ARES system provides prevention against various vulnerabilities of other existing systems.

Results and Analysis

The ARES (Anti Reverse Engineering System) System is an aggregate of static and dynamic analysis which gives it an edge over other existing systems. The existing systems have their own scope of preventing reverse engineering attacks, but the following tables provide the comparison between existing systems.

Table-1. Comparison based on Type of Analysis.

Analysis / Systems	Static Analysis	Dynamic Analysis	Both
Disassemblers	✓		
Decompilers	✓		
File Analyzers	✓		
Debuggers		✓	
Encryption		✓	
Compression		✓	
Obfuscation		✓	
Transformations		✓	
ARES System			✓

The above table illustrates how the ARES System provides both static and dynamic analysis, which makes it more wholesome and preferable.

Beyond that the following table provides a different type of analysis based on the techniques used and the existing systems. The table provides an idea on what techniques are used and which vulnerabilities are patched by existing systems.

CONCLUSIONS

The reverse engineering techniques can be wrongly used to manipulate the legitimate access of the software. This needs anti-reverse engineering techniques as a part of the software unpackaging or installing framework for evaluating the trustworthiness of the software extraction process and prevent a reasonable number of attacks as they happen. This can be a common practice in the future because of growing usage of Internet and computer access.

Table-2. Comparison based on techniques used and the systems.

Systems Compared / Prevention Techniques	Disassemblers	Decompilers	File Analyzers	Debuggers	Encryption Based Systems	Compression Based Systems	Obfuscation Based Systems	Transformations Based Systems	ARES System
Breakpoint Analysis			✓						✓
Coverage Analysis			✓						✓
Timing Analysis			✓						✓
Control/Data Flow			✓						✓
Guard Pages				✓					✓
Anti-Dumping					✓	✓	✓	✓	✓
Process Analysis				✓					✓
Stack Analysis				✓					✓

REFERENCES

[1] Dennis Yurichev. 2015. Reverse Engineering for Beginners" [Online]. Available: www.beginners.re.

[2] Teodoro Cipresso. 2009. Software reverse engineering education - Master's Thesis, San Jose State University.

[3] Shaukat Ali, Kirill Bogdanov, Neil Walkinshaw, "A comparative study of methods for dynamic reverse-engineering of state models," The Software Quality Engineering Laboratory, Carleton University, Canada, 2015, pp. 1-10.

[4] Barbara Frederiksen-Cross, Susan Courtney. 2011. Reverse Engineering: Vulnerabilities and Solutions", Johnson Laird Ltd, Forensic Software Analysis, Excerpt from PNSQC 2011 Proceedings. pp. 1-6.

[5] Teodoro Cipresso. 2009. Software Reverse Engineering Education", Masters Thesis presented to San Jose State University SJSU Scholar Works. pp. 1-122.

[6] "Software Evolution, Reengineering and Reverse Engineering", Tutorial on Software Design and Architecture, King Fahd University of Petroleum and Minerals [Online].

[7] A. Abdurazik and J. O_utt. Using UML collaboration diagrams for static checking and test generation. In International Conference on the Uni_ed Modeling Language, pages. 383{395, 2000.

[8] Aho, R. Sethi, and J. Ullman. Compilers: Principles, Techniques, and Tools. Addison-Wesley, 1986.

[9] R. Binder. Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley, 1999.

[10] L. Briand and Y. Labiche. A UML-based approach to system testing. Journal of Software and Systems Modeling, 1(1), 2002.

[11] L. Briand, Y. Labiche, and Y. Miao. Towards the reverse engineering of UML sequence diagrams. In Working Conference on Reverse Engineering, pages 57{66}, 2003.

[12] R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck. E_ciently computing static single assignment form and the control dependence graph. ACM Trans. Programming Languages and Systems. 13(4): 451{490}, Oct. 1991.