Project Specification - Group 1

Joakim Loxdal, Douglas Fischer,
Oussama Kaddami, Abdelmoujib Megzari

November 12th 2021

1 Introduction

Malware is harmful software, designed to cause some form of damage to a system without the consent of the system user. Some malware infects benign binaries to hide themselves from discovery.

To prevent this, detection software has been developed that tries to detect malware in binaries or other files by verifying signatures (hashes) [1] against a set of known signatures, or by signing the binaries cryptographically beforehand and verifying them on execution [2].

In our project we aim to implement a detection system into the OS level by storing signatures (hashes) of accepted binaries in a database and make sure that the signatures still hold before executing the binaries. We assume that the OS kernel is trusted.

2 Targeted Vulnerabilities

Malware can be injected into the system in different manners. Some types of malware such as rootkits infect the ELF files of the operating system to assure continued access, or some of the applications that monitor intrusion detection to hide themselves from the system administrator. Signing some of the critical modules in the system and doing signature verification at loading time will prevent execution of modified binaries. This can be extended to user applications and files in the files system.

The challenge is to keep an intact version of hashes that serve as references during loading time so that an attacker cannot modify this hashes as well. One efficient strategy is to use public key signing through RSA but this requires adding a crypto module into the kernel. Another strategy is the symmetric key signing, and this requires the protection of the hashes table.

3 Minimal Requirements

1. Hash some of the OS modules for which integrity is sought

2. Implement a reliable storage for known trusted hashes of binaries
3. Check signature of binaries that are executed
4. Alert the user about the potential malware and stop the execution of the binary

4 Optional Requirements

1. Use our technique on both user level binaries and kernel modules
2. Optimize the overhead due of the signature check.

References

[1] L. Litty, H. A. Lagar-Cavilla, and D. Lie, "Hypervisor support for identifying covertly executing binaries.," in *USENIX Security Symposium*, vol. 22, p. 70, 2008.

[2] A. Apvrille, D. Gordon, S. E. Hallyn, M. Pourzandi, and V. Roy, "Digsig: Runtime authentication of binaries at kernel level.," in *LISA*, vol. 4, pp. 59–66, 2004.