Senior Design Project
PlayPen

Specification Report

Ali Altaf Salemwala, Mehmet Can Altuntaş, Usama Saqib

Supervisor: Özgür Ulusoy

Jury Members: Uğur Gündükbay and İbrahim Körpeoğlu

Specifications Report
October 14, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491.
Contents

1. Introduction ........................................ 1
  1.1 Description ...................................... 1
  1.2 Constraints ..................................... 2
  1.3 Professional and Ethical Issues .................. 2
2. Requirements ....................................... 2
  2.1 Functional Requirements ........................ 2
  2.2 Non-functional Requirements .................... 2
3. References ......................................... 3
1. Introduction

A malware is any software designed to cause intentional harm to a computer or a computer system. This damage is done after it is implanted in to the victim system. Computer Economics published a report in 2007, which estimated the financial damage caused by malware in 2004 to be around 17.5 billion dollars annually. [1]

Once the malware is inside the system it is the job of the incident response team to dissect the malware, and attempt to understand and identify it. Once this is accomplished, the professional analyst may provide the malware's signature to the security apparatus, so that it may no longer propagate further into the system if it is a worm, or communicate with the Internet if it is a Trojan or a backdoor. Furthermore, the analysis reveals indispensable information about how the malware was able to penetrate into the system, so that the vulnerability can be patched.

There are millions of malwares in the wild, and more are being written everyday. Malware analysis is critical for anybody having to respond to cyber security incidents in today’s environment.

To tackle this problem there are certain products, referred to as sandboxes, that execute the malware in secured environments, and perform dynamic analysis on them in order to define the capabilities of the malware. However, most modern sophisticated malwares can detect when they are being executed in a sandbox, and change their code path in order to defeat the analysis attempt. They do this by employing a number of clever tricks, such as only executing when certain user interactions are performed, and under particular system conditions. These techniques can be very difficult to preempt and therefore make automatic analysis difficult. [2]

We attempt to tackle this problem, by creating a product that allows analysts to defeat these evasive capabilities.

1.1 Description

Our project involves a framework, which allows professional analysts to dynamically generate sandboxes with particular kinds of features, specifically designed to beat the techniques of a particular malware. For example the analyst may specify that these environment features be emulated: i) browser history, ii) left mouse click at particular intervals, iii) specific size for the SSID.

The solution is semi-automatic since the analyst will need to perform some initial analysis to figure out the malware's evasive capabilities. Once those are known, the analyst may use our framework to dynamically generate a sandbox, which will then perform behavioral analysis on the malware.

This analysis will involve the generation of "Indicators of Compromise" (henceforth referred to as IOCs). These are indicators of malware behavior. For example a ransomware may ask the kernel's cryptographic GUID in order to perform its encryption, a back-door may open a TCP connection, etc. These IOCs will then be fed to a machine-learning algo-
rithm, which will then compare the malware with a database of known malwares in order to figure out if we have discovered a zero day.

1.2 Constraints

Platforms: The program will only work on Linux systems because the sandbox requires interaction with the operating system and each operating system has its own set of system calls, therefore the product is system dependent.

Devices: Our product will only work on personal computers.

Economic: Since the resource intensive nature of machine learning, we may require external hardware. The product will also have a backend server, which would update locally hosted malware database on the client’s machine. These add extra costs to anyone running the software.

Implementation: We will use Scala and Spark for the data analytics pipeline, Python for prototyping the model, C for low-level operating system interactions.

Political: The program can be adopted or extended by political entities to find vulnerabilities that can be then used for cyber-attacks with political motivations. The software is thus politically independent and does not explicitly target or support any specific political entity.

Ethical: The zero-day exploits are not automatically reported, so the user that detects them might choose to use them for unethical gain. The program respects the privacy of the user by keeping the data private, but runs the ethical risk of having the results being exploited.

Sustainability: The program will be evolving based on new developments in malwares and sandbox evasion techniques. It will be able to grow the possible features afforded to sandboxes to remain sustainable in the market as a malware detection tool.

1.3 Professional and Ethical Issues

Malicious parties may test their malwares on our product and design it specifically to beat detection by our product. Since professional analysts will trust our product to accurately identify the capabilities of the malware, this issue may lead to some capabilities of the malware to remain undetected, which may have disastrous results.

2. Requirements

2.1 Functional Requirements

* The user should be able to select features for the sandbox.
* The user should be able to dynamically generate different combinations of these features.
* The program should be able to analyze any malware given by the user.
* The sandbox should load all the actions by the malware in a clean, human readable format.
* The program should alert any zero days discovered.
* The program should be able to do a prediction about the malware by employing a classification algorithm.
• The program should be able to tell if the file belongs to a family of known malware. For example if the file is a customized version of the known Zeus malware, the program should inform the user.

• The program should be able to archive the zero days in a local database and if the user accepts to share it, the newly-archived zero day would be sent to our servers so that the model can be updated.

• The program should be able to take the snapshot of the memory during the execution of the malware and represent this information in a helpful manner and report it to the user.

2.2 Non-functional Requirements

• The program is going to be available in English, Turkish and Urdu.

• The program should not be able to allow malware to escape from the sandbox environment.

• The program should be able to assure the data about newly archived zero days is not corrupted.

• The program should work reliably under the condition that the given file is malicious. The program may list normal programs as malicious, since it is taught to flag certain behaviors that normal software performs, but may also be performed in a malicious setting.

• The zero days discovered by the user wouldn’t be transferred to a third-part server without the consent of the user, so the program should be respectful to the user’s privacy.

• The types of "system and the user environments" the sandbox can emulate should be extensible, so that more can be added to the product, in order to improve functionality.

3. References
