WEF - Web Exploit Finder

Automatic Drive-By-Download – detection in a virtualized environment

Thomas Müller Benjamin Mack Mehmet Arziman
Hochschule der Medien (HdM), Stuttgart
{tm026,bm022,ma018}@hdm-stuttgart.de

Abstract

Much has been written about security vulnerabilities in Microsoft Internet Explorer and Mozilla Firefox. Some of these security threats are designed to execute malicious code in the browser. Known as Remote-Code-Execution-Attacks, these threats typically exploit a specific utilization of buffer overflows in an application. They are not only limited to browsers but almost all services and applications that are part of the internet or that use it as a communication platform.

We focus on internet browsers here because of two key problems. First of all, browsers are the primary user interfaces to the World Wide Web. As the rendering engine transforms hypertext into a visual presentation for human, all parts of a webpage have to be interpreted and processed further by the browser—which leads to a complex and error-prone architecture, especially in regard to mobile code (JavaScript, Java, ActiveX, XUL etc.). Secondly, the browser is arguably the most frequently used program in the family of potentially vulnerable software. In contrast to server-based software, a browser is often used by non-technical users, many of whom neither understand the risks or know possible counteractive measures. And even experts are often exposed to the risk of an attack.

In view of this, our goal was to develop a system that automatically detects and identifies malicious websites.

In addition, this system would also be able to serve as a platform for other security and sandbox-tests. One use-case is to automatically analyze various kinds of malware in a secure and easy maintainable virtualized environment.

1 Introduction

To begin with, we started by discussing some important questions and project requirements:

- How should we define the expression **“malicious”** for our project?
- What options are available for detecting malicious web content?
- What design requirements are needed for an adequate system?

Our project defines a web page that downloads, installs and executes malicious software (a virus, worms, Trojan horses, keyloggers, etc) on a client as “malicious”. We concentrate on malicious software that installs without any user interaction, making it hard to identify even for advanced users (Drive-By-Downloads).

We limited our focus in order to find web pages that actually take advantage of security bugs in the browser. At this stage, our objective was not to deal with web pages that trick the user or offer infected software as downloads. However, we’ve considered how we could add this functionality at a later date.

To find a way to detect malicious websites, you have to put yourself in the position of an attacker. What are an attacker’s goals and how can he or she achieve them? An attacker wants to compromise a user’s computer—to do this, he or she needs to change the state of the PC in some way. For instance, consider a typical scenario:

- The attacker executes his own code (shell code) with help of a buffer overflow in the browser.
- Since the functionality is very limited, given the small amount of code that can be included in the buffer overflow, he usually tries to download more code from the web and run it.

This small application is often called a “Dropper” or “Downloader”, since it downloads the actual malware to the system and includes it with some registry entries in every following system boot.

To discover such changes to a system there are at least two different options:

- *Intrusion Detection: Determine the state of the system before and after a visit to a suspicious internet page and compare both results.* You can use a list of all relevant files and registry entries coupled with their corresponding checksums to determine the “state” of a system. In this way, new or modified files can be detected easily. The key difficulties with this technique are the huge delays involved in detecting a threat as well as poor performance and scalability.
- *Rootkit: Detection using modifications to the operating system.* This technique monitors and evaluates the relevant system calls. Such changes to the operating system are not

designated and require a deeper interaction with the kernel. This procedure is also often used in rootkits and is typically called a rootkit itself.

We decided to use the rootkit technique due to its performance advantages.

In addition, to use the system as a research platform it needs to satisfy the following requirements:

- It should work automatically, requiring as little user interaction as possible.
- It should be possible to control the system remotely, such as with a web interface.
- It should be scalable and extensible.
- It should be secure, with components to ensure the system itself cannot be infected by malicious websites.

2 Limitations and Assumptions

To detect malicious webpages, we decided to use the following configuration in our test system:

- Windows XP Professional without any Service Packs
- No security updates installed
- Windows running as an Administrator
- Using Microsoft Internet Explorer 6
- Scripting and Java both activated

We chose this configuration to offer a broad attack surface. In theory, this should simulate the *worst case scenario,* but it also reflects a common configuration for many users.

3 System Architecture

To meet the requirements involved, our system architecture includes the following components:

- A **virtualization layer,** using VMware Server, to protect the system and to check multiple pages simultaneously.
- A specialized **rootkit** to modify the operating system and detect the malicious pages.
- A **Browser Control** to manage the rootkit and Internet Explorer as well as to communicate with our management console.
- A **Management Console** to configure and control the entire system.

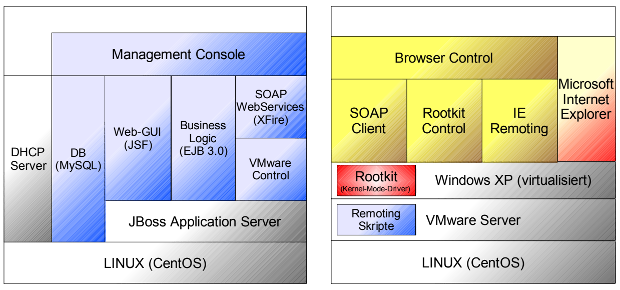

Figure 1: Architecture of the system

4 Technologies Used

For the operating system we used CentOS Linux (a fork of RHEL) on both machines. We decided to use J2EE/Java for the implementation of our management console. In addition, we used JBoss Application Server as Middleware, EJB 3.0 for the business logic and the data model as well as the Java Server Faces (JSF) for the Web-GUI.

We wrote the Browser Control and its component in C++ as a Windows-MFC application. We used SOAP for the communication protocol between the Management Console and our Browser Control.

We implemented the virtualization layer with the free server software VMware Server. The scripts for maintaining the server are a combination of C applications and “bash” scripts.

Since we decided to use “SSDT-Hooking” as the hooking-technique (a kind of redirection of system calls) the rootkit needed to have access to the protected memory of the kernel. For this reason, we implemented it as a system driver in C using the MS-DDK (Driver Development Kit).

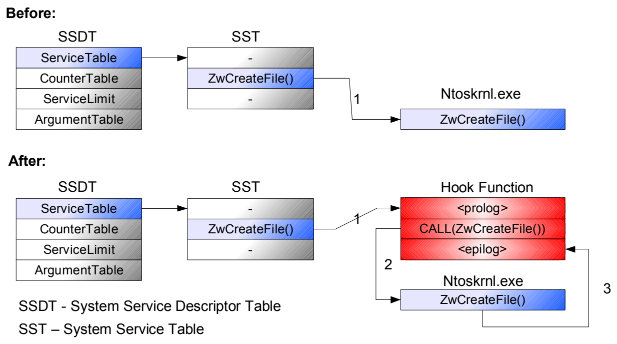

Figure 2: Functionality of the SSDT-Hooking

5 Functionality

Users can control and monitor the system via the Management Console (MC) web interface. To get started, the user enters or imports a list of URLs that the system will check. In later versions, this task can be performed by a web crawler that extracts linked URLs from webpages that are already marked as malicious.

In order to avoid jeopardizing security or interrupting performance with constant re-installations, the system visits the URLs within a sandbox. This sandbox is implemented as a VMware Server guest image. It is cloned and configured with some of our own bash scripts and the internal VMware C-API. This procedure can be repeated n-times, depending on how many virtual Windows XP instances are used to check URLs simultaneously. This also depends on the performance of the VMware host system. To guard the cloning process, the scripts notify the state of the sandbox to the Management Console which shows this on the web interface.

The sandbox images are registered and booted through the VMware Server. Once the boot process is finished an additional script creates a snapshot of the current state, copies the most recent version of our Browser Control as well as the rootkit in the sandbox and runs the Browser Control. By using the snapshot technique, the system can do a rollback once a system is infected. In contrast to deleting the sandbox and creating a new one (which takes about one minute), this process takes only a few seconds.

The Browser Control component first loads the Rootkit as a windows driver in the Windows XP kernel space. After that, the Browser Control (BC) registers itself with the sandbox and displays the IP address in the Management Console. At this point, the systems loops through the following steps:

1. BC asks the MC for the next URL via SOAP.
2. BC sends the rootkit a message to trigger the system monitor. All further relevant system calls (CreateFile, DeleteFile, Execute, etc) are then redirected and observed.
3. BC starts Internet Explorer and hands it the URL it receives.
4. Once the webpage is loaded completely or a timeout is received for this action, the browser is closed and monitoring stops.
5. Now it is time for the BC to ask the Rootkit for the results list. If the page was “clean” this list is empty, otherwise the list includes all suspicious system calls with time/date information as well as the process id (PID) of the corresponding application.
6. If the list is empty, our BC informs the MC which then marks the URL as clean in the database. If the list includes entries, these are transferred to the MC as well. If the sandbox is marked as infected, another script is called to restore the snapshot that was created at the start. After copying the BC and Rootkit and re-registering with the MC, the system returns to step one.

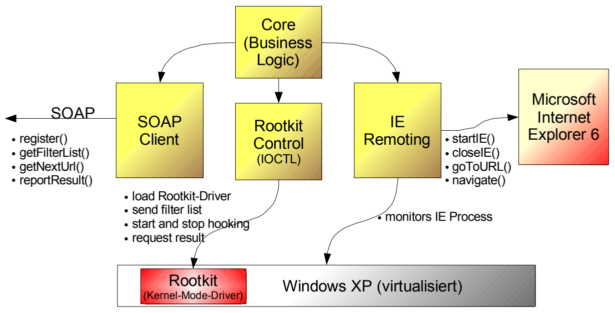

Figure 3: Design and Functions of the Browser Control - Part

6 Outlook

The system is being implemented by the three developers in their spare time, with plans to release it as an open source software - project afterwards. After the initial version is complete, the focus will shift to integrating the URL-crawler as well as further development of the rootkit. When fully implemented, the system will be able to search not only for malicious webpages automatically but also for relationships between malicious websites and their respective owners.