University of Innsbruck

Institut of Computer Science

Exploitation Techniques and Mitigations

Alex Hirsch

Patrick Ober

2016-01-31

Acknowledgement

A university course at Rensselaer Polytechnic Institut[1] held in Spring 2015 focused on *Modern Binary Exploitation*. They made their course material available on GitHub [1] under the Creative Commons Attribution-NonCommercial 4.0 International license[2]. We reused a lot of their material in this project.

We highly recommend checking them out and having a look at their material for further details.

1 Introduction

Exploiting binaries was comparatively easy in the early days of computing. Usually there were no special mitigation techniques in place trying to prevent even the most simplest exploits. This is the point in time where we will start of. First we talk about two very simple exploits, namely the Format String Exploit and the Buffer Overflow in combination with Shell Code. Note that there is a huge collection of exploitation techniques known to the public and we will thereby only look at a very small fraction of them.

But before we can introduce these two exploits, some background knowledge is required. This will be handled by the next section, which provides a short overview of the relevant components in our target architecture, the x86 platform.

After that both techniques are introduced to the reader, followed by the first mitigation technique, Data Execution Prevention (DEP). From there on we will keep on using the buffer overflow technique with some adaptations to circumvent DEP. At this point Return Oriented Programming (ROP) is introduced.

This directly leads to Address Space Layout Randomization (ASLR) the next mitigation mechanism we will discuss. Again the buffer overflow technique can be adapted to break ASLR through the use of additional information.

Since neither DEP nor ASLR provide significant protection against even this simple technique, an additional mitigation is put into place in the form of Stack Cookies.

Examples will be provided along the way to support the reader and provide some additional explanation.

Control Flow Integrity (CFI), Heap Corruption and polymorphic code will follow in a more compressed manner to communicate the main idea behind each of them.

Finally we will conclude with a word about other architectures (x86_64 and ARM) and a lookout that even languages considered secure have their own set of exploitation techniques an attacker could leverage.

1.1 Main Assumption

Throughout this work we assume that we know the target binary (and the libraries it uses). Let us show that this assumption is quite reasonable to make by looking through the eyes of the adversary. An attacker who wants to penetrate a target machine and get control over it would most likely choose the easiest path, by exploiting the weakest link. Most machines relevant to an attackers interest will run provide multiple services. For example, while the main server of a small business company may run a homemade communication server for interaction between them and their clients, it may also run a standard web server. Sending a misspelled request to the server may lead following response:

[1] http://rpi.edu/
[2] https://creativecommons.org/licenses/by-nc/4.0/legalcode

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.2.22 (Ubuntu) Server at ovinnik.canonical.com Port 80</address>
</body></html>
Connection closed by foreign host.
```

The web server tells us his exact version and since it also provides information about the operating system an attacker can easily clone the basic setup to test and tweak his exploits.

2 Platform x86

This section will teach necessary background knowledge about the target platform to fully conceive the following techniques. But first let us elaborate why x86 has been chosen in the first place.

At the time these techniques (and the related mitigations) were established, x86 was the most common platform. Since most exploits easily translate over from x86 to other architectures, especially x86_64 which very common nowadays. Also, most material found on the internet regarding this and related topics cover x86.

More detailed explanations can be found on Wikipedia[3] or the Intel Manual[4].

2.1 CPU and registers

2.2 Memory

[3] https://en.wikipedia.org/wiki/X86
[4] https://www-ssl.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

References

[1] Patrick Biernat, Jeremy Blackthorne, Alexei Bulazel, Branden Clark, Sophia D'Antoine, Markus Gaasedelen, and Austin Ralls. Modern binary exploitation, 2015. URL https://github.com/RPISEC/MBE. [Online; accessed 2015-12].