
Chapter 9. Target Exploitation

Target exploitation is one area that sets a penetration test apart from a vulnerability assessment. Now that vulnerabilities have been found, you will actually validate and take advantage of these vulnerabilities by exploiting the system in the hope of gaining full control or additional information and visibility into the targeted network and the systems therein. This chapter will highlight and discuss practices and tools that are used to conduct a real-world exploitation.

In this chapter, we will cover the following topics:

- In the *Vulnerability research* section, we will explain what areas of vulnerability research are crucial in order to understand, examine, and test the vulnerability before transforming it into a practical exploit code.
- Secondly, we will point you to several exploit repositories that should keep you informed about the publicly available exploits and when to use them.
- We will also illustrate the use of one of the infamous exploitation toolkits from a target evaluation perspective. This will give you a clear idea about how to exploit the target in order to gain access to sensitive information. The *Advanced exploitation toolkit* section involves a couple of hands-on practical exercises.
- In the end, we attempt to briefly describe the steps for writing a simple exploit module for Metasploit.

Writing exploit code from scratch can be a time-consuming and expensive task. Thus, using publicly available exploits and adjusting them to fit your target environment may require expertise, which would assist in transforming the skeleton of one exploit into another if the similarity and purpose is almost the same. We highly encourage the practice of publicly available exploits in your own labs to further understand and kick-start writing of your own exploit code.

Vulnerability research

Understanding the capabilities of a specific software or hardware product may provide a starting point for investigating vulnerabilities that could exist in that product. Conducting vulnerability research is not easy, neither is it a one-click task. Thus, it requires a strong knowledge base with different factors to carry out security analysis. The following are the factors to carry out security analysis:

- **Programming skills**: This is a fundamental factor for ethical hackers. Learning the basic concepts and structures that exist with any programming language should grant the tester with an imperative advantage of finding vulnerabilities. Apart from the basic knowledge of programming languages, you must be prepared to deal with the advanced concepts of processors, system memory, buffers, pointers, data types, registers, and cache. These concepts are implementable in almost any programming language such as C/C++, Python, Perl, and Assembly. To learn the basics of writing an exploit code from a discovered vulnerability, visit http://www.phreedom.org/presentations/exploit-code-development/exploit-code-development.pdf.
- **Reverse engineering**: This is another wide area for discovering the vulnerabilities that could exist in the electronic device, software, or system by analyzing its functions, structures, and operations. The purpose is to deduce code from a given system without any prior knowledge of its internal working, to examine it for error conditions, poorly designed functions, and protocols, and to test the boundary conditions. There are several reasons that inspire the practice of reverse engineering skills such as the removal of copyright protection from a software, security auditing, competitive technical intelligence, identification of patent infringement, interoperability, understanding the product workflow, and acquiring the sensitive data. Reverse engineering adds two layers of concept to examine the code of an application: **source code auditing** and **binary auditing** . If you have access to the application source code, you can accomplish the security analysis through automated tools or manually study the source in order to extract the conditions where vulnerability can be triggered. On the other hand, binary auditing simplifies the task of reverse engineering where the application exists without any source code. **Disassemblers** and **decompilers** are two generic types of tools that may assist the auditor with binary analysis. Disassemblers generate the assembly code from a complied binary program, while decompilers generate a high-level language code from a compiled binary program. However, dealing with either of these tools is quite challenging and requires a careful assessment.
- **Instrumented tools**: Instrumented tools such as debuggers, data extractors, fuzzers, profilers, code coverage, flow analyzers, and memory monitors play an important role in the vulnerability discovery process and provide a consistent environment for testing purposes. Explaining each of these tool categories is out of the scope of this book. However, you may find several useful tools already present under Kali Linux. To keep a track of the latest reverse code engineering tools, we strongly recommend that you visit the online library at http://www.woodmann.com/collaborative/tools/index.php/Category:RCE_Tools.
- **Exploitability and payload construction**: This is the final step in writing the **proof-of-concept** (**PoC**) code for a vulnerable element of an application, which could allow the penetration tester to execute custom commands on the target machine. We apply our knowledge of vulnerable applications from the reverse engineering stage to polish shellcode with an encoding mechanism in order to avoid bad characters that may result in the termination of the exploit process.

Depending on the type and classification of vulnerability discovered, it is very significant to follow the specific strategy that may allow you to execute an arbitrary code or command on the target system. As a professional penetration tester, you may always be looking for loopholes that should result in getting a shell access to your target operating system. Thus, we will demonstrate a few scenarios with the **Metasploit framework** in a later section of this chapter, which will show these tools and techniques.