ETH-105: Ethical Hacking

Course Length: 5 days

Course Description:

The training consists of two main sections. The first one shows the exploit writing process and the other one is about web application security. The purpose of this training is to give not just an overview but also to teach the students how to find vulnerabilities and write actual exploits. Using this knowledge, they can verify the security of applications during their ethical hacking and pentesting assignments.

The exploit writing part of the training helps students to learn how to find vulnerabilities using fuzzers for example and how to verify if the bug is exploitable or not. And if the answer is yes, it is exploitable than how to write a working exploit to take advantage of the vulnerability and how to evade different types of protections.

As there are differences between the platforms, we are giving instructions and labs to develop and exploit not just on Windows but on Linux, as well. The order of the sections and labs are in a given order and each one of them builds upon the knowledge of the previous step. So, it will happen as a process not just immediately and it helps the students to follow the training.

The second part of the training called Web application security helps the students to quickly evaluate web applications, find and expose vulnerabilities. Each section has its own hands-on lab. These practical tests show what happens behind the scenes so the students will understand what happens and not just click through an automated pentesting application. We put emphasis on the basics and technical details. We use automated tools also to speed up the processes, of course. But we lay the foundation so the trainees will understand why an attack works, how it works and what caused the problem.

Training objectives: At the end of the training participants:

- Follow the complete exploit writing process: find vulnerabilities with techniques such as fuzzing, verify whether a bug is exploitable and write a working exploit for it.
- Write stack-based buffer overflow exploits on 32-bit Windows and on 64-bit Linux, and evade protections such as Data Execution Prevention (DEP).
- Exploit format string vulnerabilities and turn the attack into a Metasploit module to make it automated and repeatable.
- Use Meterpreter in penetration testing assignments.
- Exploit the virtual function table mechanism and understand its relation to Control Flow Guard.
- Find and exploit the most common web application vulnerabilities: reflected, stored and DOM based cross-site scripting (XSS), cross-site request forgery (CSRF) and session fixation.

- Perform command injection, JWT and prototype pollution attacks, and take advantage of OAuth 2.0 authentication vulnerabilities.
- Expose logical errors such as Insecure Direct Object References (IDOR), deliver direct and indirect prompt injection attacks against Large Language Model (LLM) based applications, and exploit error based, boolean based blind and time based blind SQL injection.

Prerequisites: General Linux and Microsoft Windows system administration, basic knowledge of web applications and related protocols, basic understanding of SQL databases. Basic programming skills and understanding of programming.

Detailed Course Outline

PART I - Exploit writing

Module 1 **Stack-based buffer overflow exploit 32-bit Windows**

Stack buffer overflow is a type of the more general programming malfunction known as buffer overflow (or buffer overrun). Overfilling a buffer on the stack is more likely to derail program execution than overfilling a buffer on the heap because the stack contains the return addresses for all active function calls. This is one of the basic vulnerabilities but it's essential to understand this, as a starting point.

Module 2 **Stack-based buffer overflow exploit 64-bit Linux, DEP bypass**

DEP (Data Execution Prevention) prevents execution of shellcode on the stack. This prevents the standard buffer overflow method since the shellcode on the memory does not get executed. This would result in a SIGSEGV error. To bypass this limitation, you use pointers of things already defined and pass arguments to them, since that is still allowed. We switch to Linux as a demonstration platform, so the students can learn application hacking on a second platform as well.

Module 3 **Format string attack (fuzzing, Metasploit module writing)**

The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application. We will give an example using an actual application: how to find the vulnerability, how to write a working exploit for that type of error. At last, we walk the students through the process of writing a Metasploit module so they can learn how to make the attack more automated and repeat that much easier and faster in the future.

Module 4 **Meterpreter usage**

Meterpreter is a security product used for penetration testing. Part of the Metasploit Project and Framework, it provides enterprise security teams with the knowledge helpful for addressing vulnerabilities in the targeted application against which Meterpreter is deployed. We walk through the process of how to use it in pentesting.

Module 5 **Virtual function table exploit**

Attacker exploits the virtual function table mechanism to execute shellcode in the system. This is a more advanced attack and we will teach the students how to take advantage of that type of attack and its relation to the Control Flow Guard.

PART II - Web application security

Module 1 **Cross-Site Scripting (XSS) attacks**

XSS attacks are a type of injection in which malicious scripts are injected into otherwise benign and trusted websites. Using that kind of attack an attacker can send a script to the victim's computer and the browser will execute the script. The students will learn about the two most common types – reflected and stored – cross-site scripting attacks and the DOM based XSS attacks, as well.

Module 2 **Cross-Site Request Forgery (CSRF) + session fixation**

CSRF is an attack that forces – using a social engineering trick for example – authenticated users to submit a request to a Web application against which they are currently authenticated. CSRF attacks exploit the trust a Web application has in an authenticated user because the application cannot tell the difference between a legitimate and a forged request. The session fixation hijacks a valid user session and is often exploited using XSS attacks and HTTP header responses.

Module 3 **Command injection**

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell and executes applications. Using this technique an attacker can also attack the infrastructure as well because there's trust relationship between components in the same zone, usually.

Module 4 **JWT attacks**

JWT (Json Web Tokens) are a standardized format for sending cryptographically signed JSON data between systems. The attacks involve a user sending modified JWTs to the server in order to achieve a malicious goal. Typically, this goal is to bypass authentication and access controls by impersonating another user who has already been authenticated.

Module 5 **Prototype pollution**

Prototype pollution is a JavaScript vulnerability that enables an attacker to add arbitrary properties to global object prototypes, which may then be inherited by user-defined objects. Usually, it is exploited to be able to execute DOM based XSS attacks.

Module 6 **OAuth 2.0 authentication vulnerabilities**

OAuth 2.0 is a commonly used authorization framework that enables websites and web applications to request limited access to a user's account on another application. Vulnerabilities can arise in the client application's implementation of OAuth as well as in the configuration of the OAuth service itself. By stealing a valid code or token, the attacker may be able to access the victim's data or login as the victim user on any client application that is registered with this OAuth service.

Module 7 **Logical error: Insecure Direct Object References (IDOR)**

IDOR is a vulnerability that arises when attackers can access or modify objects by manipulating identifiers used in a web application's URLs or parameters. It

occurs due to missing access control checks, which fail to verify whether a user should be allowed to access specific data. A specific type of logical error, the "missing die" attack discussed in this section, also. That specific attack takes advantage of the programming error when the execution of code is not stopped so an attacker can execute arbitrary code after the benign one.

Module 8 **Large Language Model (LLM) attacks**

LLM attacks against web apps take advantage of the model's access to data, APIs, or user information that an attacker cannot access directly. The most common technique known as prompt injection. This is where an attacker uses crafted prompts to manipulate an LLM's output. The students will learn how to identify LLM inputs, how to deliver prompt injection attacks directly and indirectly.

Module 9 **SQL injection (Error based, Time based blind, Boolean based blind, etc.)**

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. Our hands-on labs show how and why these attacks work and what happens behind the scenes when you use automated and semi-automated tools.