Syllabus

Reverse Engineering

CYBERIUM ARENA
-SIMULATOR-

Description

Reverse Engineering is a technique used to analyze software to identify and understand its components and its flows. It is a process of understanding code infringement processes and analyzing software weaknesses. Reverse Engineers analyze systems to create system representations in another form of abstraction.

The course helps prepare for the certification exam GREM (SANS).

Target Audience

This course targets cybersecurity practitioners with experience in malware analysis, Windows forensics, and exploit development capabilities.

Pre-requisites

Linux
Forensics
Malware Analysis

Objectives

- Analyze various file formats to uncover the hidden codes within them
- Identifying control flows
- Understand Assembly
- Exploiting server, database, and application software

Module 1: Methods of Counting & Representing Information on a Computer

This stage aims to cover necessary theories and concepts which reverse engineering is based on, starting from the base structure of files and their source.

Calculation of Bases

- Hexadecimal Base
- Binary Base
- Transition Between Bases
- Transition Between Hexadecimal to Binary and vice Versa
- Numerical Actions on Numbers in Different Representations
- Negative Numbers

Module 2: Computer Systems Structure - Assembly language

During this stage, students will practice an in-depth analysis of the program codes using Assembly principles. Students will be able to recognize the effect of software and codes before their initial execution.

Assembly

- Registries
- Processor Architecture
- Portable Executable

Installing a Workspace

- Linux syscall Table
- File Descriptor
- The Connection to Files
- Start of Program Construction
- Debugging Process
- IDA

Professionalization in GOB

- Jumps & Conditions
- Manipulation on a Processor
- Loops
- Activating Number-Detonation on the Processor
- Ordering Bytes
- Maintaining Flags Mode using a Stack
- Stack
- Calling Conventions
- Build printf Functions using Assembly
- Call to Functions

Module 3: Exploitation

Students will learn about memory management and control code flows in this module while utilizing it to develop exploits.

Buffer

- Protostar
- Buffer Overflow

Writing Exploits to Bypass Protections

- Processes in Computer Science
- Pseudo-terminal
- Race Condition
- Apport Service
- How Debugger Works
- Anti-Reversing
- Return Oriented Programming (ROP)

Memory Management Policy

- W^X
- NX bit DEP
- Ret2libc
- Format String
- Overcoming the ASLR Mechanism Through the Format String Attack
- The Process of Adding the Addresses to a Written Code

Memory Management

- How a Process Gets Memory from the System
- Heap Overflow

Preparing a Windows Workspace

- Visual Studio
- OllyDbg

Exploitation Over the Internet

- Buffer Overflow Over the Internet
- Tracer Browser Detection
- Fuzzing
- SPIKE
- Debug Using OllyDbg to Restore Crash
- Shellcode
- Manually Create Shellcode
- Create Shellcode Using Metasploit

Bad Characters

- Encoding
- From Python to Metasploit Mixins
- Slmail
- Immunity Debugger
- Mona.py