Offensive Security Methods Prof. Dr. Bernhard Jungk
Hochschule Albstadt-Sigmaringen SS 2022

Exercise 4 - Binary Exploitation

General Hints

- The exploitable source file `exploitable.c` can be downloaded from Ilias here: `https://elearning.hs-albsig.de/goto.php?target=file_395340_download&client_id=HSALBSIG`.
- You have to deactivate most countermeasures for the exploits to work in a simple way as shown in the lecture slides.
- Furthermore, it is easier to attack a 32 bit target (x86 architecture) instead of attacking a 64 bit target (x86_64 architecture). You can use either directly the Metasploitable VM or compile your source code using the `-m32` option of gcc.
- To be able to debug and inspect the code in `gdb` and using `objdump`, you have to compile the source code using the `-g` option to include debugging symbols in the executable.
- If you are unable to figure out the correct addresses to jump to quickly, you should try to add a simple NOP sled.

Tools

If not already familiar with the tools, you need to get familiar with the following tools:

- From the GNU toolchain `gcc`, `gdb`, and `objdump`.
- As assembler you can use either `nasm` or the GNU assembler `gas`.

Exercise 4.1 - A Basic Stack Overfow Attack

Exploit the error with `gets` in `exploitable.c` to jump to the function `void hidden(void)`.

Hints

- The input to the executable can be supplied using the standard input `stdin`, e.g., using a pipe. The easiest way is to prepare your payload is outside the debugger (e.g., as exploit.bin) and to use `r < exploit.bin` as command to run the exploit.
- Follow the approach presented on the lecture slides and adapt it to use `stdin` instead of command line arguments.
- You have to figure out the target address of the hidden function and overwrite the return address on the stack to jump to that function.

Success Condition

You are successful, if the `printf()` called by the hidden function is invoked.

Exercise 4.2 - Spawning A Shell With Shellcode

Exploit the error with `gets` in `exploitable.c` to spawn a shell (`/bin/sh`) using your shellcode and the system call `execve()`.

Hints

Calling a system call on the x86 architecture works as follows:

- The `execve()` documentation is available at `https://man7.org/linux/man-pages/man2/execve.2.html`.
- The system call number is put into register `eax`.
- The official system call numbers for the x86 architecture can be found here: `https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl`. Note, that the system call numbers are different for other architectures.
- The first parameter is put into `ebx`, second parameter in `ecx`, third parameter in `edx`, fourth parameter in `esi`, fifth parameter in `edi` and the sixth parameter in `ebp`. The result is returned in `eax`.
- If the all registers are filled, the system call is executed with `int $0x80`
- Details how to call a system call can be found at `https://en.wikibooks.org/wiki/X86_Assembly/Interfacing_with_Linux`.

The assembly code must be turned into machine code using an assembler such as `nasm` or `gas`. Using `objdump` it is possible to access the machine code which is necessary to craft the final shellcode.

Success Condition

You are successful, if a shell is started. In your report you should proof your success by executing some commands in the shell. Furthermore, you should test, if your exploit also works with activates ASLR.

Exercise 4.3 - Spawning A Shell With ret2libc

Exploit the error with `gets` in `exploitable.c` to spawn a shell (`/bin/sh`) using the glibc function call `system()`.

Answer the following question: What is the major difference and benefit for the attacker between Exercise 4.2 and 4.3?

Hints

Calling a library function works differently from executing a system call.

- The documentation for `system()` can be accessed here: `https://man7.org/linux/man-pages/man3/system.3.html`
- The function `system()` takes exactly one argument, which must be put on the stack.
- You need to either find the string `/bin/sh` somewhere in the memory space of the executable (e.g., in an environment variable), or you have to put the string on the stack yourselves.
- Finding the string `/bin/sh` works with

Success Condition

You are successful, if a shell is started. In your report you should proof your success by executing some commands in the shell. Furthermore, you should test, if your exploit also works with activates ASLR.

Exercise 4.4 - Spawning Meterpreter

Exploit the error with `gets` in `exploitable.c` to spawn a bind shell generated using `msfvenom` from Metasploit.

Hints

The general approach is very similar to the shellcode injection in Exercise 4.2. However, instead of executing a system call, the payload generated with `msfvenom` must be injected and executed. You can then connect using netcat to the TCP port.

There are many ways to create a payload using `msfvenom`. This is the recommended way for this exercise:

- Use `msfvenom -help` and Internet searches to find out more about the payload generation process. It is also useful to use the `-l` option of `msfvenom` to find out more about the available payloads, encoders, etc.
- You should use a `shell_bind_tcp` payload appropriate for your platform (linux) and architecture (x86). This specific payload will open a simple TCP port, which you can connect to using netcat.
- Specify a port using, e.g., by using LPORT=8888 as option to `msfvenom` to select the TCP port the bind shell will listen on.
- Connecting using netcat works with `nc TARGET_IP TARGET_PORT`. If successful, all commands you type (and send using enter) are executed on the target host. As proof you should list a directory such as `/bin`.

Success Condition

You have successfully connected to the remote shell using netcat at least once and executed some commands on the remote shell.