(SWE3025, Spring 2025) Homework Assignment #1

Instructor: Hyungjoon Koo (kevin.koo@g.skku.edu)

Your Name : _____

Student ID : _____

The project must be submitted on an individual basis. The following states several rules for both submission and grading policy. Please read the descriptions of each exercise carefully before you make a final submission.

- You must submit a single zipped file via `iCampus`. The submission file name should be your student ID with the file extension of `.zip`.
- The submission must contain four files: a single PDF report that answers all questions with proper descriptions, and three python scripts with `.py` file extension. The script name should be `exp1.py` that produces `badfile`, `exp2.py` for shellcode injection, and `exp3.py` for the `return to libc` attack.
- Each script will be run with `python3.7` or higher version for evaluation. Thus, there is no need to submit `payload` data. Note that any answer without well-description may be degraded.
- The environmental setup must be done via Docker.
- The deadline for the homework is midnight (11:59 PM) on Apr. 17th (Thursday), 2025.
- Last but not least, cheating (copying, retyping, outsourcing, submitting copies of others) will bring about failing this course.

Docker

Docker is an open-source platform that enables the deployment and management of applications using containerization technology. A Docker container packages software into a standalone unit, including the code, runtime, system tools, and libraries necessary for execution—essentially replicating a complete system environment. This guarantees that users to provide the same environments.

Docker Installation on Ubuntu in Windows WSL

For the installation of Docker in Windows WSL (Windows Subsystem for Linux), you may want to check out this reference [3]. Listing 1 briefs the steps for Docker installation in Windows WSL.

Listing 1: Docker installation in Windows WSL

```
# Update apt packages in WSL
$ sudo apt update -y
$ sudo apt upgrade -y

# install necessary packages
$ sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

# add docker GPG key and storage
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share
    /keyrings/docker-archive-keyring.gpg

$ echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://
    download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.
    list.d/docker.list > /dev/null

# update a docker package list
$ sudo apt update -y

# install a docker
$ sudo apt install -y docker-ce docker-ce-cli containerd.io

# add docker privilege to user account
$ sudo usermod -aG docker <USER>

# execute a docker image
$ sudo service docker start

# test a docker after re-connecting
$ docker run hello-world
```

Docker Installation on Ubuntu

Installing Docker in Linux is relatively simple as the following Listing 2.

Listing 2: Docker installation in Ubuntu

```
# install a docker
$ curl -fsSL https://get.docker.com/ | sudo sh
sudo usermod -aG docker <USER>

# test a docker
$ docker run hello-world
```

Building a Docker Image

With the provided `Dockerfile` file and the `file` directory, you can build your own image as in Listing 3. To disable ASLR, you must grant permissions to the Docker image with `--privileged` option when running it. The image has disabled ASLR by default. A 'tag name' is an identifier to access to the image.

Listing 3: Building a docker image and running it.

```
# build a docker image
$ cd <docker image path>
$ docker build . -t <tag_name>

# run a docker with a privileged mode
$ docker run -it --privileged <tag_name>
root@d8a574ada8a3:/$ cat /proc/sys/kernel/randomize_va_space
0
root@d8a574ada8a3:/$ ls -l
total 16
drwxrwxrwx 1 root root 4096 Apr  3 07:58 1.bof
drwxr-xr-x 1 root root 4096 Apr  3 07:58 2.shellcode
drwxr-xr-x 1 root root 4096 Apr  3 07:58 3.R2L
drwxr-xr-x 1 root root 4096 Apr  3 07:58 peda
```

You can find general help in the official documentation [1] by Docker. Once you generate the docker image for the homework assignment, you will have three tasks to complete.

Question 1.

(25 Points) Stack Overflow

As discussed in class, a stack overflow occurs (in the presence of a vulnerability) when a well-crafted exploit fills up a buffer, followed by overflowing into the space beyond the buffer. Your task is to write `exp1.py` to create a bad file by leveraging a buffer overflow on a vulnerable program, `stack.c` in Listing 4, to execute the `authenticated()` function.

```c
// gcc -fno-stack-protector -m32 -no-pie -g -o stack stack.c
...
void login(char *str) {
    char buffer[BUF_SIZE];
    unsigned int *framep;

    strcpy(buffer, str);
}

void authenticated() {
    printf("Congratulations!\n");
    printf("You passed the first level\n");
    exit(0);
}

int main(int argc, char **argv) {
    char input[1000];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    int length = fread(input, sizeof(char), 1000, badfile);

    printf("Can you beat me?\n");

  login(input);

    printf("Unfortunately, return successfully :(\n");
    return;
}
```

Listing 4: Source code for Question 1.

In the `1.bof` directory, you can find the skeleton code for `exploit.py`, `stack.c`, and the `stack` executable from the source. When you execute `stack`, it will give you a segmentation fault (Listing 5).

```
user@4729d3ceb14d:~/1.bof$ ls
exploit.py  stack  stack.c
user@4729d3ceb14d:~/1.bof$ ./stack
Segmentation fault
```

Listing 5: First Program Execution.

With `badfile`, you should be able to get the following message as in Listing 6. Write the `exploit.py` code so that it can produce the following output, which creates the `badfile`. Note that rename the python code with `exp1.py`. Besides, you should submit a report about how you achieve such results.

```
user@4729d3ceb14d:~/1.bof$ sudo python3 exploit.py
user@4729d3ceb14d:~/1.bof$ ls
badfile  exploit.py  stack  stack.c
user@4729d3ceb14d:~/1.bof$ ./stack
Congratulations!
You passed the first level
```

Listing 6: Running the First Executable with `badfile`.

Question 2.

(35 Points) Shellcode Injection

Now, our goal is to spawn a shell by injecting a shellcode into a vulnerable process's memory and executing it. The shellcode is a small piece of code written in a machine language. Note that the pwntools [2] simplifies an exploitation process. With a careful look at the `stack.c` code (Listing 7), your task is to write `exp2.py` for spawning a shell. You may want to execute basic Linux commands, such as `ls` and `id`.

```c
// gcc -z execstack -fno-stack-protector -m32 -no-pie -g -o stack stack.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifndef BUF_SIZE
#define BUF_SIZE 100
#endif

void dump_to_buffer()
{
    char buffer[BUF_SIZE];
    /* This also can be checked by gdb */
    printf("Address of buffer[] inside bof():  0x%.8x\n", (unsigned)buffer);

    gets(buffer);
}

int main(int argc, char **argv)
{
    printf("Could you get shell for me?\n");
    printf("I need your help\n");
    dump_to_buffer();

    printf("Unfortunatly, return successfully :(\n");
    return;
}
```

Listing 7: Source code for Question 2.

In the `2.shellcode` directory, you can find the skeleton code for `exploit.py`, `stack.c`, and `stack`. When you execute the second program, you can see the following output (Listing 8).

```
user@4729d3ceb14d:~/2.shellcode$ ./stack
Address of buffer[] inside bof():  0xffffd6cc
|
```

Listing 8: Second Program Execution.

Write `exploit.py` for successfully spawning a shell (Listing 9). Note that rename it with `exp2.py`. Besides, you should submit a report about how you achieve such results.

```
user@4729d3ceb14d:~/2.shellcode$ python3 exploit.py
[+] Starting local process './stack': pid 270
exploit.py:10: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://
    docs.pwntools.com/#bytes
  p.recvuntil("bof():  ")
BOf addr :  0xffffd6bc
[*] Switching to interactive mode
$ ls
exploit.py  stack  stack.c
$ id
uid=1000(user) gid=1000(user) groups=1000(user)
```

Listing 9: Running the Second Executable with Shellcode Injection.

Question 3.

(40 Points) Return to Libc

We move onto the `ret2libc` technique as discussed in class. In short, it borrows desirable functions from the `libc` library to bypass non-executable memory or data execution prevention (DEP). With a careful look at the `retlib.c` code (Listing 10), your task is to write `exp3.py` for spawning a shell.

```c
// gcc -mpreferred-stack-boundary=2 -z norelro -no-pie -fno-pic -g -fcf-protection
    =none -fno-stack-protector -m32 -o retlib retlib.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifndef BUF_SIZE
#define BUF_SIZE 12
#endif

const char shell[] = "/bin/sh";

void login() {
    char buffer[BUF_SIZE];

    scanf("%s",buffer);
}


int main(int argc, char **argv) {
   printf("Try to get Shell!\n");
   printf("Can you get the system(%s)?\n", shell);

   login();

   return;
}
```

Listing 10: Source code for Question 3.

In the `3.R2L` directory, you can find the skeleton code for `exploit.py`, the `disable_aslr.sh` script, the `retlibc.c` source, and the `retlibc` executable built from the source. When you execute `retlibc`, it will give you the following output as Listing 11.

```
user@4729d3ceb14d:~/3.R2L$ ./retlib
Try to get Shell!
Can you get the system(/bin/sh)?
|
```

Listing 11: Third Program Execution.

Your task is to write `exploit.py` for successfully spawning a shell (Listing 12). Note that rename the python code with `exp3.py`. Besides, you should submit a report about how you achieve such results.

```
user@4729d3ceb14d:~/3.R2L$ python3 exploit.py
[+] Starting local process './retlib': pid 288
[*] Switching to interactive mode
Try to get Shell!
Can you get the system(/bin/sh)?
$ ls
disable_aslr.sh  exploit.py  retlib  retlib.c
$ id
uid=1000(user) gid=1000(user) groups=1000(user)
```

Listing 12: Running the Thrid Executable After the Successful `ret2lib` Attack.

References

[1] Docker. dockerdocs. https://docs.docker.com/reference/cli/docker/.

[2] Pwntools. Pwntools. https://github.com/Gallopsled/pwntools-tutorial/tree/master.

[3] André Silva. Installing Docker on WSL 2 with Ubuntu 22.04. https://gist.github.com/dehsilvadeveloper/c3bdf0f4cdcc5c177e2fe9be671820c7.