In the name of God

Sharif University of Technology
Department of Computer Engineering

CE 815: Secure Software Systems

Mehdi Kharrazi *Aban 26^{th}, 1404*

Homework 1*

1 Get Familiar with Git

You should submit all your HWs at S4lab's Git distributed revision control system named Tarasht. To access Tarasht, your account and personal repositories will be emailed.

1.1 Git Config

In order to use Git, we recommend using a Linux machine or using a Linux virtual machine. Git uses a username to associate commits with an identity, and your username is like **ce815-041-student-id**. Using following commands, you can specify Git configuration settings with the git config command. One of the first things you should do is to set up your name, username and email address:

```
git config –global user.name ”Your name”
git config –global user.username ”ce815-041-student-id”
git config –global user.email ”you@sharif.edu”
```

1.2 SSH Key

You can generate a new SSH key (or use an existing SSH key) for authentication, then add it to your Tarasht account to modify your repositories without entering passwords every time. Using following commands, you can generate a new pair of ssh key:

```
ssh-keygen -t rsa -b 4096 -C ”you@sharif.edu”
eval ”$(ssh-agent -s)”
ssh-add  /.ssh/id_rsa
```

After key generation, log in at Tarasht, open your account's setting and add your public key. Your public key should start with **ssh-rsa**.

1.3 HW Submission

You can find your personal repo at Tarasht, and its name is exactly like your username; therefore your personal repo URL address is `https://tarasht.ce.sharif.edu/ce815-041-students/ce815-041-student-id`.

First clone your repository with the following command:

*Acknowledgement: This homework was developed by Zahra Fazli and Mohammad Hadadyian

```
git clone https://tarasht.ce.sharif.edu/ce815-041-students/ce815-041-student-id.git
```

For each HW, you should create a new folder with the name of HW id, i.e., **HW1**. We need a pdf file as HW's report, codes and related data to be added to this folder.

After adding your HW's directory, use the following command to submit your data:

```
git add HW1
gitt commit -m "Finished HW1"
git push
```

1.4 HW Handouts

All handouts can be found at handouts repository `https://tarasht.ce.sharif.edu/ce815-041-students/ce815-041-handouts`. You should first clone this repository:

```
git clone https://tarasht.ce.sharif.edu/ce815-041-students/ce815-041-handouts.git
```

2 Simple Memory Corruption Attack

Your first assignment is easy Format String and ROP attacks. We have a vulnerable binary program, which for the sake of simplicity, we have also released its C source code. You should first find the vulnerability and then write a simple script (bash, Python,Perl, etc.) to smash the stack and make it spawn a shell. The Aleph One tutorial, **Smashing The Stack For Fun And Profit**, and also Stanford's **Exploiting Format String Vulnerabilities** are your friend.

The vulnerable program and its source code is available at handouts repository.

- **prog_vuln1:** this program is a 32bit executable that has no security feature, so with crafted input you can reach the shell.
- **prog_vuln2:** this program is a 32bit executable that has NX (Non-eXecutable) security feature on, so instead of putting the shell code into stack you should use ROP attack and find proper gadgets.

2.1 Tools

Gdb is an appropriate tool for solving this assignment. To get familiar with gdb, you can visit this tutorial. There is also a powerful extension for professional gdb work called gef (gdb Enhanced Features), which is useful for solving challenges and for deeper study; see its repository at this site .

You can use tools such as gdb and objdump to locate return addresses.

You can use tools like pwntools, PEDA, and ROPgadget **only** to help automatically find gadgets

2.2 Delivery

- You must submit a written report that explains your script and documents the entire procedure step-by-step.

  Any report that does not list the steps in detail will receive zero points.

- You must also submit the script itself and include clear instructions on how to run it.
- Learn the procedure thoroughly: on the day you hand in the assignment you will be asked to perform the exploit on a similar vulnerable program on a machine without internet access.

3 CVE Analysis and Exploitation

Find and exploit a real-world stack-based memory corruption vulnerability from a notable project.

3.1 Requirements

Select a CVE from 2022 onwards that involves stack-based memory vulnerabilities (buffer overflow, stack smashing, etc.) from a well-known open-source project.

Important: Before starting your work, submit your chosen CVE to the TA for verification. The CVE must not have a publicly available Proof of Concept (PoC) or exploit. Each student must work on a different CVE.

Your submission must include:

1. **CVE Identification**: Specify the CVE number, affected project, and version.
2. **Vulnerability Analysis**:
   - Pinpoint the exact vulnerable code line(s) in the original project
   - Explain technically why this code is vulnerable
   - Describe the security implications
3. **Proof of Concept**:
   - Create a working program that contains the vulnerable logic
   - The program must be functional, not just the vulnerable snippet
   - It can be extracted from the real project or a simplified version, but must demonstrate the actual vulnerability
4. **Exploit Development**:
   - Write a complete working exploit script
   - Document your exploit development process
   - Include instructions to compile and run both the vulnerable program and exploit

3.2 Delivery

- A comprehensive report covering all requirements above
- Source code of your vulnerable program
- Your exploit script with clear execution instructions
- **Important**: You must demonstrate your exploit live during submission. Be prepared to explain your approach and answer technical questions about the vulnerability and your exploitation technique.

Note: Simply copying CVE descriptions or existing exploits without understanding will not receive credit. You must demonstrate deep technical understanding of the vulnerability and exploitation process.

4 Challenges

In this section, two remote pwn challenges are provided. For each challenge, you will receive:

- A Dockerfile to set up a consistent testing environment on your personal computer
- The binary file for the challenge

The Dockerfiles are provided for your practice and local testing. However, you will only receive points by successfully exploiting the challenges on the target remote servers.

4.1 Setting Up Local Environment

To ensure the Docker environment runs properly on your system, make sure your system architecture is x86. After installing `docker`, navigate to each challenge directory and use the following commands to build and run the environment:

```
./dockerbuild.sh
./dockerrun.sh
```

You can then connect to your local instance using:

```
nc localhost [local_port]
```

If needed, you can use `docker exec` to access the Docker container for inspecting and debugging.

Before attempting exploitation, examine the security features of each binary using tools like `checksec`.

To start the exploit, since the program's code is not available to you, it's better to examine it using debugging or decompiling tools. Note that in this section of the exercise, you are not provided with a shellcode, and you must achieve the goal using the program's functions.

4.2 Challenge 1: Student Registry

Connect to the remote server using:

```
nc 65.109.185.79 2003
```

When you connect, the program will greet you and ask for your name and grade. Your task is to find the vulnerability and exploit it to capture the flag on the remote server.

4.3 Challenge 2: Todo Manager

Connect to the remote server using:

```
nc 65.109.185.79 2004
```

This challenge runs a simple todo application. You can interact with it to manage your tasks. Explore the application's functionality, identify its weakness, and exploit it to retrieve the flag.

4.4 Delivery

For each challenge:

- Submit a detailed report explaining the vulnerability you discovered and how you exploited it.
- Include your exploit script with clear instructions on how to run it.
- Document each step of your exploitation process.
- **Note:** Flags must be captured from the remote servers, not from local Docker instances.
- Flag format is CE815{xxxx}.