Lab 4 - Buffer Overflow 1

ⓘ **Course & Instructor Info**

Course Name: SWE 4504 - Software Security Lab
Instructor: Imtiaj Ahmed Chowdhury, Lecturer, CSE, IUT

Objective

After finishing the lab, students will have acquired the skills necessary to proficiently analyze C programs. They will be able to identify buffer overflow vulnerabilities with ease, thanks to their comprehension of the x86 function calling convention and stack layout. Moreover, they will be equipped to generate Python exploit scripts that can activate a shell on the system running the vulnerable program. They would also learn use of debuggers and reverse engineering frameworks.

Overview

Buffer overflow is one of the most dangerous vulnerabilities of all which can lead to arbitrary remote code execution resulting in full system takeover. Its imperative for all students of Software Engineering that they know how to spot and also avoid this vulnerability in the software they will be developing. They should also know the implications of this type of vulnerability in any existing software.
In this lab, we'll be working with **32-bit x86 binaries** containing buffer overflow vulnerability. We'll use `gdb` to dynamically analyze a given binary and also use `Ghidra` to decompile a given binary. We'll be particularly using an advanced version of GDB, GEF, an open source debugger. Ghidra is a reverse engineering software developed by NSA through which we'll be able to see both disassemble assembly and decompiled source code from a given binary executable.
We'll be demonstrating exploitation of buffer overflow vulnerabilities in C programs by overwriting local variables on the stack, calling arbitrary functions and executing shellcode.

Essential Commands

To run 32-bit ELF executables on 64-bit linux, install the following:

```bash
$ sudo apt install gcc-multilib
```

Check address of function inside gef:

```
gef➤  p <function_name>
```

For any other commands that you might need to solve the tasks, just google your query.

Lab Task (complete within the lab)

Marks: 15

Download the C source file `vulnA.c`, corresponding binary `vulnA` and `flag.txt`. Your task is to call the `win` function and print the success message.

Tip: If there's two input-taking function (*gets*, *fgets*, *scanf* etc.) one after another and the first one terminates at `\n`, then you can pass payload to both input taking functions with a one-liner exploit like following:

```
$ python -c 'print "A"*10 + "\n" + "B"*10' | ./vuln
$ python3 -c 'import sys; sys.stdout.buffer.write(b"A"*10 + b"\n"
+ b"B"*10)' | ./vuln
```

Here, first set of 10 A's will go into first input-taking function and second set of 10 B's into second input-taking function.

Lab Assignment (complete before next lab and submit in Google Classroom)

Marks: 25

Download the corresponding binary named `vulnA<group_id>` and the **fake** `flag.txt` from the Google Drive folder attached with the assignment. The same binary will be running on a remote server on `142.93.223.6` at port `69<Group_ID>`. For example, if your group ID is `15` then the binary will be running on port `6915`. You can then connect to the binary like `nc 142.93.223.6 6915`.
Your task is to analyze the given binary using *GEF* and *Ghidra* and exploit it locally to call the **win** function. After the exploit works on your local machine then you run it against the binary on remote server. If the exploit script is successful then you'll get the actual **flag**.

N.B. You are not given the source code for the assignment. Figure out how the binary works using GEF and Ghidra.

Submission

Submit your **exploit script/exploit command** for the assignment task after renaming it like `hackA<group_id>.txt` and the **flag** that you got after successful exploitation on remote server.
Also submit your exploit script/exploit command for the lab task.