COMP 30080: Computer Systems

2021 / 2022

Assignment 4

Pavel Gladyshev

UCD School of Computer Science,
University College Dublin,
Belfield, Dublin 4.

Introduction

There are five assignments in total. Assignments should be submitted using CS Moodle via the 'Assignment submission' links. The assignment deadlines are provided in Moodle. Roughly speaking, one assignment is to be submitted every two weeks of term. Moodle will allow submissions up to 2 weeks late. However, late submissions incur penalties according to UCD policy unless a medical certificate or similar is submitted to the lecturer or the UCD Science Programme Office.

Assignments do not have to be done during lab times. However, Demonstrators are only available during lab times.

Assignment 1 is not assessed. Students should compare their solutions with the model solution provided. All assignment marks and feedback will be available in Moodle.

For all assignments:

- For each assignment, submit an accompanying report (.doc or .pdf) which includes a brief explanation of what you have produced, screenshots showing verification of the submitted program, any workings and the answers for written work questions. Use your student number and assignment number as the file name, e.g. 12345678_ass1.doc
- Make sure that your name and student number are visible in the assembly files, circuits, and report.
- If necessary, scan (or photograph) any handwritten work into a file for submission. Submit the graphics file (.gif or similar) or include the scan in the report.
- Submission is via Moodle. Moodle requires a single file that should be a .zip of all individual files. Use your student number and assignment number as the file name, e.g. 12345678_a1.zip

PLEASE NOTE, THE REPORTS MUST BE IN WORD OR PDF FORMAT, THE ASSEMBLY PROGRAMS MUST BE IN ASM (TEXT) FORMAT AND THE SUBMISSION MUST BE IN ZIP FORMAT. OTHER FORMATS OR CORRUPT FILES WILL NOT BE MARKED

The marking scheme for most questions is: 0=nothing done; 1-3=partial working; 4=working; 5=working and elegant.

Assignment 4: Exploiting and hardening vulnerable C programs

In this assignment we will study buffer overflow attack: its causes, implementation, and the defense against it. Buffer overflow attacks were brought to public attention by Elias Levi (a.k.a. "Aleph0"), the CTO and co-founder of SecurityFocus, in his landmark paper "Smashing The Stack For Fun And Profit.

It was the first public, accessible, step-by-step introduction to buffer overflow vulnerabilities. It is by no means secret information: that paper currently has 1316 citations according to Google Scholar.

PLEASE NOTE that any **deliberate** attempt to carry out an **unauthorized** buffer overflow attack against computer system owned or operated by another physical person or legal entity is a **criminal offence** in most jurisdictions including in Ireland. At the same time, it is essential for you as future IT professionals to understand the mechanisms behind such attacks and to ensure that the software you develop is protected against them. That is why we included this exercise in COMP 30080.

1) Read about buffer overflow attack

Familiarize yourself with the "Smashing The Stack For Fun And Profit" article, which can be found at

https://inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

Although the attack principles described by Levi are applicable to most computer architectures, he uses Intel x86 CPU assembly language to demonstrate his ideas.

I have adapted his ideas to MIPS Linux environment and provided you with an example of how a simple buffer overflow vulnerability can be carried out against a vulnerable MIPS program. Please study this example alongside the pre-recorded presentation from November 18th, 2020.

2) Download and install GNU C compiler and related utilities for MIPS little endian architecture

You will need to use GNU C compiler and binary utilities for MIPS32 Little Endian (MIPSEL) CPU to compile the vulnerable C program and to develop your shellcode. Therefore, you will need a virtual machine (or a physical PC) running recent version of Ubuntu Linux. All tasks specified in this document had been verified on the **mips-tools** virtual appliance.

To install GNU C compiler for MIPS32 Little Endian CPU, QEMU userland emulator, and multi-architecture GNU debugger. Start a terminal window and run the following commands:

```
sudo apt-get install qemu-user qemu-system-mips

sudo apt-get install gcc-mipsel-linux-gnu gdb-multiarch
```

For additional information about C calling convention please refer to Chapters 7 of MIPS Assembly Language Programmer's Guide (https://csmoodle.ucd.ie/moodle/mod/url/view.php?id=1617) .

To use GNU debugger, open two terminal windows and navigate each of them to the sub-directory containing your program

1. Start MIPS simulator in one window using command:

   `qemu-mipsel -g 1235 -L /usr/mipsel-linux-gnu/ yourProgram`

   here -g 1235 option specifies TCP port waiting for GNU debugger connection.

2. Start GNU Debugger in the other window using command:

   `gdb-multiarch yourProgram -ex "target remote :1235" -ex "set arch mips"`

3. Once GDB starts switch to the interactive "Text User Interface" mode by entering command

   `tui enable`

   If the source code and debugging information is available, enable display of program source code and CPU registers by entering commands:

```
layout regs
```

alternatively, if the source code and/or debugging information is not available, display program disassembly and CPU registers by entering commands:

```
layout asm
layout regs
```

4. Step through the program **s**ingle **i**nstruction at a time using command

   ```
   si
   ```

   or **s**ingle line (of the source code) at a time using command

   ```
   s
   ```

   to run (continue) program until completion (or the next break point) use command

   ```
   c
   ```

You can set breakpoints using command 'b' please refer to this and other commands' description in the official GDB user manual (https://sourceware.org/gdb/current/onlinedocs/gdb/) and various online tutorials.

3) Download and experiment with provided examples

Download `a4-files.zip` and `example.zip` files provided alongside this assignment and **unzip** them somewhere in your virtual machine.

Study the example of shellcode (`example-sc.s`) provided in the `example.zip` alongside the vulnerable server program, client software and shellcode compilation scripts. Compile the client and server programs using commands

```
mipsel-linux-gnu-gcc --static -fno-stack-protector -g3 -o server server.c
```

```
mipsel-linux-gnu-gcc --static -fno-stack-protector -g3 -o client client.c
```

Start server program using command

```
qemu-mipsel server 8001
```

or simply

```
./server 8001
```

where 8001 is the TCP port that the server is to listen on.

You can give server commands using client program as follows:

```
echo -n "version" | ./client 127.0.0.1 8001
```

where 127.0.0.1 and 8001 are the IP address and TCP port that the server program is listening on.

Assemble shellcode using provided shell script `assemble`

```
bash assemble example-sc
```

Make sure that you DO NOT specify .s extension at the end of the shellcode assembly file. If all goes well, file `example-sc.bin` will be created. You can review its content by disassembling it using provided shell script `disassemble`

```
bash disassemble example-sc.bin
```

If your setup is correct, you should be able to exploit the vulnerability of the server by feeding the shellcode file to the server via the client software using the following command:

```
cat example-sc.bin | ./client 127.0.0.1 8001
```

where 127.0.0.1 and 8001 are the IP address and TCP port that the server program is listening on. This should cause the server program to report an error message and run Linux 'ls' command, which is the deviant behavior induced by the shellcode.

4) Assignment questions

In this assignment you will need to find solutions for two questions.

Q1. Locate folder `q1/` in the `a4-files.zip` It contains a slightly modified version of the example server in which commands are NULL-terminated strings rather than binary blobs. In order to exploit that program, your shellcode MUST NOT contain any zero bytes.

Your task for Q1 is to modify the shellcode program `q1-sc.s` provided in `q1/` so that once assembled it does not contain any 0x00 bytes and can be used to exploit Q1 server as described in the previous section.

To find a solution for Q1 you will have to use MIPS instructions in innovative ways. Here are some **HINTS** for you to consider:

- Although the default NOP instruction has code 0x00000000, you can use any other instruction that does not affect program behavior in place of NOPs.
- .data and .text section sizes must be a multiple of 16 bytes, otherwise the assembler and linker might automatically pad it with 0x00000000s. Check disassembled code.
- `execve()` system call requires its argument strings `cmd`, `amd_arg1`, `cmd_arg2` and the array `exec_args[]` to be NULL-terminated (i.e. strings `cmd`, `amd_arg1`, `cmd_arg2` must be terminated with 0x00 byte and `exec_args[]` with 0x000000 word). The binary shellcode file cannot contain any 0x00 bytes, so you will need to find a way to add these NULL bytes at runtime.
- The starting address of `exec_args[]` must be a multiply of 4. Assembler may automatically prepend `exec_args[]` with some 0x00 bytes otherwise. Watch out for that.
- Read-up about the optional parameter of MIPS syscall instruction (you can use it like “syscall 0x1”). Find out a way to use it!
- More than likely you will need to debug your shellcode with gdb. Please refer to the pre-recorded presentation on how to do it.
- Extend server.c with a line that prints the length of the received data. If your shellcode binary does not contain any 0x00 bytes, the amount of data received by the server should match the size of the shellcode binary file.

Once your modified shellcode is working, include a description of your solution in the report and upload your final version of `q1-sc.s` as the answer to question 1.

[14 Marks]

Q2. Locate folder `q2/` in the `a4-files.zip` Identify and fix the vulnerability in `websrv1.c` program, describe the vulnerability and your method of fixing it in the report. Save the corrected server program as file `q2.c` and submit it as the answer for this question.

[6 Marks]

(20 marks in total)