Secure software development and web security
Homework 2

R. Absil

Academic year 2025 - 2026

This homework requires students to develop attacks related from the third to sixth chapter of the course, that is, in particular, remanence-based leaks and stack buffer overflow.

You are expected write a PDF report to answer questions by *explaining* the manipulations you use, at least with screenshots, and submit it on the page of the course on Nov. 26 by 23:59 at the latest[1] alongside the code you wrote in an archive under the `.zip`, `.7z`, `.tar.gz` or `.tar.xz` format[2].

Question 1 (3 mks)**.** In the binary named "secret-str", find the secret string hidden[3].

Question 2 (4 mks)**.** Consider the `C11` code listed below. This code implement some form of protection against buffer overflow. Explain *two* different ways of bypassing this protection in the particular case of buffer overflow.

```c
int secret; //will be initialised with a random number in the main function

void stuff(char* str)
{
  int guard = secret;

  char buffer[12];
  strcpy(buffer, str);

  if(guard != secret)
  {
    printf("Stack smashing detected. Terminating program.\n");
    exit(1);
  }
}
```

Question 3 (3 mks)**.** In the binary named "check-pwd", find a buffer overflow vulnerability and and how to exploit it to bypass password protection.

For the two following questions, it is assumed you have an old-enough 32-bits linux environment, either booted directed from a drive, or through a virtual machine. We advise to use an Ubuntu 12.04 32-bits image, that you can download from the archive[4].

[1] No delay of any kind shall be tolerated.
[2] A `.rar` archive is *not* accepted.
[3] You will know what it is when you find it.
[4] https://old-releases.ubuntu.com/releases/12.04/ - Last accessed on November 6, 2025.

Question 4 (10 mks)**.** In the binary named "check-pwd-crit", find a buffer overflow vulnerability to make it print "Critical function" without making it crash. This binary was produced from a file "check-passwd.c" under the elf32 format using the command

- `gcc -o check-passwd-crit -m32 -z execstack -fno-stack-protector check-passwd.c`

Turn both of these binaries into set-uid programs.

Question 5 (20 mks)**.** Given the binary named "root-me-1", turn it into a set-uid program and find a buffer overflow vulnerability in order to log as root. This binary was produced from a file "greeter.c" under the elf32 format using the command

- `gcc -o root-me-1 -m32 -z execstack -fno-stack-protector greeter.c`

Question 6 (10 mks)**.** Given the binary named "root-me-2", turn it into a set-uid program and find a buffer overflow vulnerability in order to log as root. This binary was produced from a file "greeter.c" under the elf32 format using the command

- `gcc -o root-me-2 -m32 -fno-stack-protector greeter.c`

Question 7 (20 mks)**.** Given the binary named "root-me-3", turn it into a set-uid program and find a buffer overflow vulnerability in order to log as root. This binary was produced from a file "greeter2.c" under the elf32 format using the command

- `gcc -o root-me-3 -m32 -z execstack -fno-stack-protector greeter2.c`