# Computer Security
# Coursework Exercise CW3
# Software Security and Binary Attacks

Margus Lind

## Introduction

This coursework focuses on software security and binary exploitations. The environment for this coursework is yet again a VirutalBox VM. The VM is set up such that it does not keep any state between launches - if anything goes wrong or the VM does not work you can simply turn it off and on again, or if that does not help, delete it and re-run the setup script.

You have access to the VM hosting the tasks. Further, a shared folder is set up between the host machine and the guest VM. This will enable you to access your solutions on both. If you keep your solutions in the shared folder, you will not lose any work should you need to re-initialise the VMs.

The target VM has a number of user accounts set up, which you should be aware of:

**task1** holds the first target executable in it's home folder

**task2** holds the second target executable in it's home folder

**task3** holds the third target executable in it's home folder

**task4** holds the fourth target executable in it's home folder

**task5** is running the service for the fifth task

**student** is the account you have access to in order to mount your attacks, the password is `student`

Each of the tasks has you facing with a vulnerable executable. In the first four cases you must attack the executable by providing malicious command line arguments. For the last task, you face a vulnerable web service instead.

In all cases you will have access to the source code. You should read the source code of the targets and understand what you are attacking.

Each task hides a 128 character hexadecimal secret, which your submissions must retrieve. The secret is indeed an unsalted SHA-512 password hash, however cracking these is not part of the exercises.

All tasks carry an equal weight of 20 marks, out of a total 100 for the coursework.

## Setup

For your convenience we have provided the following script which sets up the VM for you.

`/group/teaching/cs/cw3/setup.sh`

Running the script will create a VM in VirtualBox under your account. The VM uses the disk images stored in the class group directory. The VM disk itself will not consume your disk quota. Any changes within the VM are non-persistent.

Binary exploits are generally flaky to say the least. Small changes in the platform might cause your attacks not to work. You must ensure that your solutions work on DICE, on VirtualBox version 5.1.28

r117968 (Qt5.6.1) provided. Your submission will be assessed on an identical untouched VM - test that it works. See further below for more details on assessment.

# 1 Simple buffer overflow

The first task is a bit of a warm-up. You must provide an attack script that makes `/home/task1/task1` believe that you have entered the correct password by overwriting the relevant variable. The target will then reveal the secret it protects.

# 2 Jump to function

However, usually you will need to work a bit harder to exploit a system. Provide an attack script that will force the execution flow to reach the `read_secret` function in the target executable.

# 3 Defeating a canary

Often, many obstacles stand in your way. This target script has been compiled with stack canaries enabled. Overcome this, load your shellcode into memory, and recover the contents of the file `/home/task3/secret.txt`. Do not mount a return-to-libc type attack.

# 4 Return-to-libc

Now it is time to use a slightly more advanced technique. Mount a return-to-libc type attack on the target executable.

# 5 Vulnerable web service

For the last part, you must abuse the vulnerable web service running on port 6666 TCP. The service allows clients to read the files on an emulated file system. Each of the files is tagged with a flag stating if it is public or not. Files listed as public will be served back on request, however others will yield an error message instead.

You must submit a script that recovers the secret hash contained within the file `Secret` on the targeted file server and prints it out.

If you crash the service, you can restart it by rebooting the VM. To save you from this annoyance, you may want to copy the script over and run it yourself under your own account in the VM. This way you have better control over launching the script. However, don't forget to ensure your submission works on the original service.

# Submission and marking

You are required to submit attack code for each of the tasks. Your submission will be tested on an identical "clean" VM. The launch scripts will be executed, and your submission marked based on the success of your attacks. Help us give you full marks by following the requirements below:

- You will need to submit one directory per task.
- The directory must be named the same as the relevant user account, e.g. `task1`.

- Each submitted directory must contain a `attack.sh` script with the correct hashbang and the executable flag set.
- You are allowed to use other scripts, or tools that are present on the VM: for example, you could submit a C source file that is compiled by commands in the `attack.sh` script and executed against a vulnerable target. However, we will **only** execute the `attack.sh` script - any further logic must be completely automated.
- The `attack.sh` will be executed in the user `student` home directory `/home/student`. That means, if you submitted the directory `task1` with files `task1/attack.sh` and `task1_exploit.c`, these files will be copied into `/home/student` as `/home/student/attack.sh` and `/home/student/task1_exploit.c`. Only the `attack.sh` file will be executed.
- Your attack must follow the approach outlined in the task descriptions above - if in doubt, ask. Alternative approaches may be given 0 marks.
- Submissions compromising or rooting the VM will not be accepted - stick to the tasks at hand (and then in your free time feel free to play around). You are not allowed to submit attacks that target the platform itself.
- You must refer to the target executable by it's absolute path.
- The `attack.sh` must set up and execute your attack, as well as sanitise the output and print out **only** the required secrets. **For each task, your `attack.sh` script must output only the 128 hex character SHA512 secret hashes.** The secret hashes we will use for marking your submissions will be different - so your solution cannot simply print the hash, it must actually carry out the attack.
- All tasks carry an equal weight of 20 marks, out of a total 100 for the coursework.

You need to submit by the deadline of ~~**16:00 on Monday, November 27th**~~ **10:00 on Wednesday, November 29th**.

You're reminded that late coursework is not allowed without "good reason", see

`http://www.inf.ed.ac.uk/teaching/years/ug3/CourseGuide/coursework.html`

for more details about this, and the procedure to follow if you must submit late. In particular, if you have a good reason to submit late, please use the ITO support form `http://www.inf.ed.ac.uk/admin/ITO/support/index.html` to make a request.

## Good Scholarly Practice:

If you use any content in your submissions that is not created by yourself for and during this assessment, it must be clearly cited.

Please remember the University requirement as regards all assessed work for credit. Details about this can be found at:

`http://www.ed.ac.uk/academic-services/students/undergraduate/discipline/academic-misconduct`

and at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).