This is the second laboratory session out of three in the Secure Programming course. Each lab session will examine some software vulnerabilities, including their exploits and repairs. The lab is split into several exercises.

Lab exercises are part of the delivery of the Secure Programming course. Your work in labs does not contribute towards the final mark for the course, but you are expected to take part. You will need to do this to gain the background needed for the upcoming coursework assignment that does contribute towards the final mark, as well develop knowledge that will be required in the exam.

- **Guided introduction.** The lab will start with a short introduction, giving some hints about the exercises. Please arrive on time for this.

- **Tools and techniques.** There are some pointers to useful tools in the exercises and in the appendix at the end of this handout.

- **Working together is encouraged during labs.** We want to foster a supportive learning environment. Students who have prior knowledge or expertise are especially encouraged to work with others. Collaborating on the exercises may help you to think more deeply about the problems by discussing different aspects, as well as sharing existing knowledge.

- **Course staff will be on hand.** We will be here to discuss your progress and help with solving the problems. Detailed help will only be available during the timetabled labs.

- **Submit answers.** There are checkpoint questions in each exercise which you can answer to measure your progress. You may submit answers using the electronic submission mechanism or simply discuss them in labs. Submitting answers will allow us to give feedback at the next session.

As the work is intended to be completed in the lab and not take more of your time, there will be a short deadline for submissions. For this lab session, submissions will be accepted until **4pm, Monday 6th March**.

For an online copy of this handout, visit [http://www.inf.ed.ac.uk/teaching/courses/sp/2016/labs/lab2/](http://www.inf.ed.ac.uk/teaching/courses/sp/2016/labs/lab2/).
Virtual machinery

For this lab we are providing some code and a virtual machine for you to use. The VM has two users, user and root. The passwords are the same as their usernames. To install the VM, you should use a virtual disk file stored in the scratch space on your machine, and let Virtual Box use it from there.

To do this you will need to change some settings in VirtualBox. First set Settings→General→Default Machine Folder to

```
/tmp/[your username]
```

Then import the appliance from the file:

```
/afs/inf.ed.ac.uk/group/teaching/module-sp/SecureProgramming-2.ova
```

The machine is set to use NAT. Once started you can either use the console window, or SSH in via your local machine over port 2222 (recommended).

```
ssh -p 2222 user@localhost
```

Lab exercises are in the folders in /home/user/ on the VM. To compile the examples, just type `make`.

Beware that /tmp is local to your current workstation, it is not backed up. So you should save any work that you do inside the virtual machine (edited source files, etc) in your home directory. We also recommend keeping a lab notebook with notes of what you have done during the exercise.
Question 1: An Authentication Program

Inside the folder /home/user/SQLi there is a Java program that does authentication by checking a user password stored in the users.db.

To authenticate, a user runs the Login program passing it their username and password. The Login program tries to remove SQL metacharacters before passing the arguments to a Java program which queries the database. You should assume that java can not be run directly by an attacker, it is only invoked by the Login front-end program. (However, as a secure programming investigator, you are free to invoke the program as you like).

Checkpoint 0. Investigate the Login program and login as one of the users.

There are several problems with the implementation. Your task is to identify and fix the problems.

You might find it helpful to start by investigating the main CWE for SQL injection, CWE-89. One of the Detection Methods it recommends is to used automated static analysis to detect SQL injections. You can try doing this by running the findbugs program on the class file.

NOTE: Your testing and investigating steps may corrupt the database users.db, we have provided a reset script to restore the original state of the database. You can run this by executing the command /home/user/SQLi/resetdb.sh. In case you have managed to corrupt even more than the database, you will need to re-import the virtual machine (or make a snapshot to restore).

Checkpoint 1. What does the output of findbugs tell you?

Checkpoint 2. Which part of the authentication program is vulnerable to SQL injection and how can an attacker exploit it?

Inside the Login script you will see that there is an attempt at removing SQL metacharacters. The attempt is flawed, however, as the escaped characters can still be inserted into a username. Investigate how to do this.

Checkpoint 3. Why isn’t the removal of quotes and semicolons through sed adequate to protect against SQL injection?

Now try to fix the code against this vulnerability and any other injection vulnerabilities you can see in Login and Login.java. Before editing, make backup copies, so you can run diff command (diff <oldfile> <newfile>) to make a patch showing your edits. Run make to recompile the Login.java to make Login.class.

Checkpoint 4. Give your patch to repair Login and Login.java and explain how it works.

After you have repaired the authentication program, show us your patch. Then we will provide you with a password to unlock some exploits. Now examine and run these exploits against your fixed program to see if you have successfully hardened the authentication program.

Checkpoint 5. Examine the exploits and explain how they work. What design flaws or vulnerabilities are abused by the exploit?

Checkpoint 6. Did your patch stop the exploits? If not explain why and provide an updated patch that does.
Question 2: Another SQL injection

Inside the folder /home/user/SQLi-2 there is a Java program that allows users to create an account and then login and change their password.

To create a user, run the Login program with the add command and the user’s credentials. To login, run the Login program with the login command and the user’s credentials.

Checkpoint 0. Check you understand the Login program. What are the users already in the database and what attacks against them can you imagine?

There are several problems with the implementation. Your task is to identify and fix the problems.

This time we are using PreparedStatements to construct our SQL queries; these improve naked queries as they allow SQL elements to be parameterized and appropriately escaped for whichever database is used.

NOTE: Your testing and investigating steps may corrupt the database users.db, we have provided a reset script to restore the original state of the database. You can run this by executing the command /home/user/SQLi-2/resetdb.sh. In case you have managed to corrupt even more than the database, you will need to re-import the virtual machine (or make a VirtualBox snapshot to restore from).

Checkpoint 1. When you run the findbugs program on the class file it shows there is an SQL injection problem despite the use of a prepared statement. Why?

Checkpoint 2. Fix the code and provide a patch for Login.java.

Follow the same procedure as with the previous exercise. After you have repaired the program, show us your patch. Then we will provide you with a password to unlock an exploit. Now examine and run these exploits against your fixed program to see if you have successfully hardened the program.

Checkpoint 3. Describe how the exploit works.

Checkpoint 4. Verify your patch stops the exploit. If not, make a new patch that does.
Question 3: Linkers (Advanced)

In this exercise we will explore how the dynamic linker works and how we can alter its behaviour at runtime.

Inside the folder /home/user/LD_PRELOAD there is a C program Vulnerable that checks whether a password correct. This is a simplified example, but the code could form part of a DRM system or a login program. You can run the program by ./Vulnerable <Password>.

There are some problems with the way the code is written and compiled. Your task is to identify the problems and suggest fixes. We do not expect you to code up fixes (unless you want to) but you should discuss and evaluate the effectiveness of any fixes you suggest.

Linux systems typically use the ELF format for binary programs. Inside the ELF program header is the path of the dynamic interpreter used to load the program and the libraries needed to run it.

1. Run the readelf program on Vulnerable with the -l flag.
2. The ldd program\(^1\) can be used to display the libraries dynamically linked with a binary. Use this program to find out the libraries the program requires.

Checkpoint 1. What program is used to interpret the ELF file Vulnerable?

When running the program we can also ask the dynamic linker to say what it is doing. We can ask it to say how it looks for the libraries, how it resolves the function symbols and how the program runs with its constructors and destructors, for example.

Look at CWE-426: Untrusted Search Path. It recommends using black box detection methods to investigate search paths.

3. Run the program with LD_DEBUG=libs (it is an environment variable) to see how the libraries are loaded.
4. What problems can you see with the way the code given that it is dynamically linked?
5. How could it be fixed?

Checkpoint 2. What is an untrusted search path? How it related to dynamically linking of library?

Checkpoint 3. How could the Vulnerable program be fixed to avoid the search path exploit?

After you have thought about this, have a look at the exploits in the exploit/ directory, and consider whether your proposed fixes were adequate.

Checkpoint 4. Describe what each of the provided exploits does, how likely you consider it to be an achievable exploit and under what circumstances.

Checkpoint 5. How could you prevent each of the attacks?

Checkpoint 6. (optional) Modify exploit-ld.sh so it attacks the SHA1 call.

\(^1\)Actually ldd is a wrapper program to the dynamic linker. The dynamic linker uses environment variables to modify how it works and whether to display debugging information. See man 8 ld.so if you are interested in these.
Submission instructions

Download a copy of the text file checkpoints.md from

http://www.inf.ed.ac.uk/teaching/courses/sp/2016/labs/lab2/checkpoints.md

and edit it to insert your answers. Submit it with the command:

submit sp lab2 checkpoints.md

We will give some feedback about the submissions at the next lab session.

David Aspinall and Arthur Chan, February 2017. Thanks to Joseph Hallett for some of the exercises.