University of Dundee
AC31012/AC51042

Dr. Saša Radomirović

Information Security
Assignment 1

Deadline: 9pm on Friday the 18th of October, 2019

Introduction

In cybersecurity, a *supply chain attack* refers to the tampering with a software or hardware component during its development or manufacture in order to subvert the end product's security. For example, an attacker might compromise a software developer's github account in order to introduce security vulnerabilities in the developer's software projects. Once the produced software is deployed, the attacker can attack the developer's customers by exploiting the planted vulnerabilities.
See the following links for some high-profile examples of recent supply chain attacks.

- `https://en.wikipedia.org/wiki/Supply_chain_attack`
- `https://arstechnica.com/information-technology/2019/06/google-confirms`
- `https://arstechnica.com/information-technology/2019/08/the-year-long-rash`
- `https://arstechnica.com/information-technology/2018/10/two-new-supply`

Assignment Specification

In this applied cryptography assignment you will, as a team (AC31012 students) or as an individual (AC51042 students), be first implementing **in C++** a secure password login (authentication) procedure and then additionally a password login procedure with a covert backdoor.
A part of your Assignment 2 mark will be determined by other teams' (in-)ability to find the backdoor in the subverted procedure that you are developing here. Your team will therefore also have to consider operational security: Ensure that no one outside of your team has access to your code and your ideas.
In part 2 of this assignment only the teams' backdoored login procedures will be distributed as source code. You may assume that the other teams will not have access to your secure login procedure nor its source code, so you do not need to worry about code similarity between your secure and your backdoored code.
This assignment can be solved in Ubuntu on the QMB Lab PCs. You may use any other computer and operating system, but it is then your responsibility to ensure that your code also compiles on an Ubuntu configuration as found on the QMB Lab PCs.

Password File Format

Your log in procedure must be able to read files that contain zero or more lines containing a username and a **SHA256**-hashed password in the format

`username:SHA256-hashed-password`

For example, a password file might look like this:

```
root:5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
alice:5ef2c394d5b63e4175cd331c74c8453c3e36eb8f47f6d648397ff6c1314fd705
```

For the above example password file, you can verify on the command line that the root user's SHA256-hashed password is "password" by typing
`echo -n "password" | openssl sha256`
Similarly, you can verify that user "alice" has the password "mushroom".
The openssl development library (preinstalled on QMB's Ubuntu desktops) provides the necessary functions to compute sha256 hashes. You will have to `#include "openssl/sha.h"`.
As a starting point, you may want to read the following stackoverflow post:
`https://stackoverflow.com/questions/2262386/generate-sha256-with-openssl-and-c`

Functionality

Your secure and your backdoored login procedure **must** satisfy the following requirements:

- R1 The login procedure must work with the password file format described in the previous section.
- R2 It must include the **authlib.h** header file and use the two functions defined therein.
- R3 It must call the function `void authenticated(std::string u)`, where `u` is the username, whenever a user enters a correct username and password pair.

Only your secure login procedure **must** satisfy the following additional requirements:

- R4 It must call the function `rejected(std::string u)` if an invalid username and password pair was entered.
- R5 It must not call `authenticated(std::string u)` unless a correct username and password pair for username `u` was entered.

It is up to you to decide whether your secure and backdoored login procedures offer the user one or more attempts to log in before rejecting and exiting.
Your secure and backdoored login procedures may offer additional functionalities, which may help disguise backdoors, **but shorter** backdoored login submission **will receive more marks**, see the marking scheme for details.
The files `authlib.h`, `authlib.cpp`, a skeleton `login.cpp` file and a `Makefile` are provided on MyDundee. You must not modify `authlib.h` or `authlib.cpp`. You cannot assume that `authlib.cpp` will be implemented in the same manner when your code is tested.

Submission

Submission deadline: 9pm on Friday the 18th of October, 2019.
Submit by email

- a `zip` file named after your **group**
- with the subject line "[your group name] - Assignment 1"
- to me (`s.radomirovic@dundee.ac.uk`)
- **and include all group members in the CC field.**

The zip file must contain:

1. A file `login.cpp`. This is the secure password login procedure. Your `login.cpp` program must
   (a) satisfy requirements R1–R5,
   (b) compile without warnings when the flags `-Wall -pedantic -Wextra` are used.
   (c) hash the submitted passwords with openssl's sha256 hash function.
   (d) The source code must be commented.
2. A file `login-subverted.cpp`. This is the password login procedure with a backdoor. Your backdoor must allow you to login as root or any other user on the system without knowing their passwords and also
   (a) satisfy requirements R1–R3,
   (b) compile without warnings when the flags -Wall -pedantic -Wextra are used.
   (c) hash the submitted passwords with openssl's sha256 hash function.
   (d) The source code must be commented, but the comments may be misleading.
3. A file `report.pdf`. This PDF file documents the vulnerability in your backdoored login procedure. It must be no more than 1 page, list the team name and team members, and address the following points:
   (a) Steps to trigger the vulnerability. How can an attacker log in to a system without knowing the root user's or some other user's password?
   (b) What are the bugs/vulnerabilities in the code?
   (c) Why do you think are your bugs/vulnerabilities difficult to detect?
   (d) Optional, not part of the page limit: A statement concerning the team members' contributions in case that not all team members have made equal contributions.
4. A `Makefile`. This file compiles both your secure and your subverted login procedures.

Marking Scheme

See the file *Assignment 1 Specification* on MyDundee for the marking scheme.