cs3235:2: - Laboratory #2 for September 4th, 2019

The first two parts of this laboratory could be done in pairs. Each of you should submit the answer, identifying your lab partner. However Parts 3 and 4 are individual submissions, counting towards your (individual) homework. Discuss the questions with your lab partner, but do not copy answers.

1 Lab 2, part 1: Exploring a protocol

The following protocol is supposed to provide mutual authentication between Alice and Bob, who share a secret key k. Both Alice and Bob keep track of nonces they have received, and will not accept a repeated *incoming* nonce. Alice and Bob can have multiple sessions/connections open at the same time.

$A \rightarrow B$	:	$A + n_a$	I am Alice, and here is a nonce
$B \rightarrow A$	:	$n_b + E(k, n_a)$	OK, here is another nonce, and your nonce encrypted with k
$A \rightarrow B$	:	$E(k, n_b)$	OK, here is your nonce encrypted with our shared key k

The protocol has three steps. Firstly, Alice sends a random one-time-value (a nonce) to Bob. Bob returns another nonce, and an encrypted version of the one that Alice just sent him, using a secret key that only Alice and Bob know. Finally, Alice returns an an encrypted version of the nonce that Bob just sent him. At the end of the protocol, both Alice and Bob are supposed to believe that the person who sent them the nonce knows the shared key.

Discuss this with your lab partner. Can you think of a way that this protocol could be attacked by Harry, even though Harry does not know the shared key? Harry will be able to pretend to be Alice, and will send Bob all the correct messages, even though Harry will never know the key. Alice will not be involved (she will send and receive no messages at all).

Once you understand how the attack is done, access the cs3235 grading website to enter in a brief description of how Harry can pretend to be Alice:

`https://hugh.comp.nus.edu.sg/cs3235/lab2/gradeslab2-1.php`

Again, you will only see your own explanation of the attack, not everyone else's. If you do this with a lab partner, put both of your IDs in the submission.

2 Lab 2, part 2: SEED Linux capabilities

This is a modification of a part of another SEED lab.

There are several ways for user-level programs to interact with the capability features in Linux; the most convenient way is to use the `libcap` library, which is now the standard library for capability-related programming. On the pre-built Ubuntu Virtual Machine, `libcap 2.21` is already installed.

For this lab, you need to get familiar with the following commands that come with `libcap`:

- `setcap`: assign capabilities to a file.
- `getcap`: display the capabilities that carried by a file.
- `getpcaps`: display the capabilities carried by a process.

In a capability system, when a program is executed, its corresponding process is initialized with a list of capabilities (tokens). When the process tries to access an object, the operating system will check the process capabilities, and decide whether to grant the access or not.

In operating systems, there are many privileged operations that can only be conducted by privileged users. Examples of privileged operations include configuring network interface cards, backing up all the user files, shutting down the computer and so on. Without capabilities, these operations can only be carried out by superusers, who often have many more privileges than are needed for the intended tasks. Therefore, letting superusers to conduct these privileged operations is a violation of the Least-Privilege Principle.

Privileged operations are very necessary in operating systems. All `setuid` programs involve privileged operations that cannot be performed by normal users. To allow normal users to run these programs, `setuid` programs turn normal users into powerful users (e.g. root) temporarily, even though the involved privileged operations do not need all the power. This is dangerous: if the program is compromised, adversaries might get root privilege.

Capabilities divide the powerful root privilege into a set of less powerful privileges. Each of these privileges is called a capability. With capabilities, we do not need to be a superuser to conduct privileged operations. All we need is to have the capabilities that are needed for the privileged operations. Therefore, even if a privileged program is compromised, adversaries can only get limited power. This way, the risk associated with a privileged program can be lowered quite significantly.

Capabilities has been implemented in Linux for quite some time, but they could only be assigned to processes. Since kernel version `2.6.24`, capabilities can be assigned to files (i.e., programs) and turn those programs into privileged programs. When a privileged program is executed, the running process will carry those capabilities that are assigned to the program. In some sense, this is similar to the `setuid` files, but the major difference is the amount of privilege carried by the running processes.

We will use an example to show how capabilities can be used to remove unnecessary power assigned to certain privileged programs. First, let us login as a normal user, and run the following command:

```
% ping www.google.com
```

The program should run successfully. If you look at the file attribute of the program `/bin/ping`, you will find out that `ping` is actually a `setuid` program with the owner being root, i.e., when you execute `ping`, your effective user id becomes `root`, and the running process is very powerful. If there are vulnerabilities in `ping`, the entire system can be compromised. The question is whether we can remove these privileges from `ping`.

Let us turn `/bin/ping` into a non-`setuid` program. This can be done with the following command (you need to login as the root[1]):

```
# chmod u-s /bin/ping
```

Note: Binary files like `ping` may located in different places in different distributions of Linux, use `which ping` to locate your `ping` program.

Now, run `ping www.google.com`, and see what happens. Interestingly, the command will not work. This is because `ping` needs to open a RAW socket, which is a privileged operation that can only be conducted by `root` (before capabilities were implemented). That is why `ping` has to be a `setuid` program. With capabilities, we do not need to give too much power to `ping`. Let us only assign the `cap_net_raw` capability to `ping`, and see what happens:

```
$ su root
# setcap cap_net_raw=ep /bin/ping
# su normal_user
$ ping www.google.com
```

To find out more about capabilities, use `man capabilities`. The `ep` setting in the command above sets the CAP_NET_RAW bit in both the "effective" (e) and "permitted" (p) capability sets.

Please turn `/usr/bin/passwd`, a `setuid` program into a non-`setuid` program, without affecting the behavior of the program. You should use the minimum capabilities that still allow the command to work.

Once you have succeeded in doing this, access the cs3235 grading website and enter in the list of capabilities that you used, and a clear description of the commands you used, the steps you followed:

`https://hugh.comp.nus.edu.sg/cs3235/lab2/gradeslab2-2.php`

Again, you will only see your own answer, not everyone else's. If you do this with a lab partner, put both of your IDs in the submission.

3 Lab 2, part 3 (individual): Design Principles (a)

A common technique for inhibiting password guessing is to disable an account after three consecutive failed login attempts. You could argue that this is an example of fail-safe defaults, because by blocking access to an account under attack, the system is defaulting to a known, safe state. Argue this both ways - that is, find arguments both for and against this position. Put your answer here:

`https://hugh.comp.nus.edu.sg/cs3235/lab2/gradeslab2-3.php`

You can complete this in the evening, or tomorrow, but must be finished before midnight, 6th September 2019. If you are outside of NUS, you may have difficulty accessing the website. If this happens, use the NUS VPN to connect.

[1] Check laboratory 1 for the root passwords.

4 Lab 2, part 4 (individual): Design Principles (b)

The postscript language is an interpreted language, used to describe page layout for printers. Among its features is the ability to request that the interpreter execute commands on the host system. Describe a danger that this feature presents when the language interpreter is running with administrative or *root* privileges. Explain how the principle of least privilege could be used to reduce this danger. Put your answer here:

`https://hugh.comp.nus.edu.sg/cs3235/lab2/gradeslab2-4.php`

This must be finished before midnight, 6th September 2019. If you are outside of NUS, you may have difficulty accessing the website. If this happens, use the NUS VPN to connect.

5 Extra fun

A few of you are having a bit of fun exploring my web site, seeing what can be done to hack it. It may not be that hard. If you want practice at this sort of activity, I would recommend this web site:

`http://www.hackthissite.org/`

It is a reasonably safe site to go to, where you can hone your awesome powers, trying out a graded series of attacks. You will need to get an account there, and should use a working email address. They dont seem to spam. If you do have a bit of fun on hackthissite, at the end of this course, show me your final grade there, for a vast quantity of money :)