Instituto Superior de Engenharia de Lisboa
MEIC, MEET, MEIM
Cibersecurity
Module 2: Exercise 1

1 Enunciado

In 2014 a vulnerability was identified in the *shell* application on several Linux, Unix and OS X systems, which became known as *Shellshock*. One way to exploit this vulnerability was for the attacker to remotely execute commands or programs on the vulnerable machine. The following sections are based on the *Shellshock* [3] vulnerability lab. For the following points take into account the background information on how to exploit the vulnerability (see Section 2), and use the Ubuntu 16.04 version of the SEED labs [1, 2].

1. Check that the lab environment is correct by following point 2.2 of the SEED Lab (*2.2 Task 2: Setting up CGI programs*).

2. Perform point 2.3 and briefly describe how an attacker with remote access to the vulnerable machine can pass arbitrary data to the execution context of a *bash shell* via the Apache server's *Common Gateway Interface* (CGI). Explore data passing through a *browser* and `curl` tool.

   Note: To edit the CGI program there are several text editors available in the VM, namely *gedit* and `vi`.

3. Launch two virtual machines (attacker and victim) and explain the command to get the following file from the victim machine: `/var/www/SQLInjection/safe_home.php`. In the file, identify the username and password used by the *web* application to access the database.

4. Would it be possible to get this way the contents of the file `/etc/shadow`, which contains the hash of the users' *passwords*? Justify.

2 Introduction to Shellshock Vulnerability

This appendix is an adaptation of Chapter 3 of the book *"Computer Security - A Hands-on Approach"* [4].
One of the most used command lines on Unix/Linux family systems is the *bash shell*, located in `/bin/bash`. In *bash* functions can be defined, as shown in the following example. Note that in the SEED Labs VM the vulnerable bash program is named `bash_shellshock`.

```
$ foo() { echo "Inside␣function"; } # defines function 'foo'
$ declare -f foo                    # shows the code of function 'foo'
foo() {[...]}
$ foo                               # executes function 'foo'
Inside function
$ unset -f foo
$ declare -f foo
$
```

Functions defined in the parent process can be exported and thus used in the child process, as shown in the example:

```
$ foo() { echo "Inside␣function"; }
$ declare -f foo
$ export -f foo
$ bash              # executes /bin/bash in a child process
(child)$ declare -f foo
(child)$foo
Inside function
```

The Shellshock vulnerability is related to the possibility of passing functions from a parent process to a child *bash* process. In addition to the method presented above (which is not a vulnerability) the other way to pass functions is through the explicit definition of environment variables. When the parent process exports an environment variable whose value is a user-defined *string*, with the definition of a function, the variable's value is interpreted and transformed into a function in the child process, as shown in the example:

```
$ foo='() { echo "Inside function"; }'
$ echo $foo
() { echo "Inside function"; }
$ declare -f foo
$ export foo
$ bash
(child)$ echo $foo
(child)$            # in the child process there is no variable 'foo' ...
(child)$            # ... but there is a function with name foo
(child)$ foo ()
{
  echo "Inside function"
}
$ foo
Inside function
```

When the child process runs `echo $foo` nothing is displayed because, during the creation of this process, if an environment variable starting with `()` is found, its value will be analyzed to transform it into a function with the same name. During this analysis, due to a programming error, the commands are executed after the second brace, as shown in the following example:

```
$ foo='() { echo "Inside function"; }; echo "Hi!";'
$ echo $foo
$ () { echo "Inside function"; }; echo "extra";
$ export foo
$ bash
Hi!          # the extra command is executed
$ echo $foo
$
$ delcare -f foo
foo ()
{
  echo "Inside function"
}
```

Note that in the second method of passing functions, the parent process need not be *bash*. Any process that wants to call the *bash shell* and pass it a function, just has to define it in an environment variable beforehand.

Referências

[1] `https://seedsecuritylabs.org/Labs_16.04/Documents/SEEDVM_VirtualBoxManual.pdf`

[2] `https://seedsecuritylabs.org/labsetup.html`

[3] `https://seedsecuritylabs.org/Labs_16.04/Software/Shellshock/`

[4] Wenliang Du, Computer Security: A Hands-on Approach, 1ª Edição, CreateSpace Independent Publishing Platform, 2017