Chapter 3: Shellshock Attack

Problems

3.1. When a shell variable containing a shell function definition is passed down to a child process as an environment variable, what is going to happen to the function definition?

3.2. Assume a Bash program defines a shell function, exports it, and then starts a child process that also runs Bash. Please explain how this function defined in the parent Bash becomes a function in the child Bash.

3.3. Write a Bash function definition that tries to exploit the Shellshock vulnerability.

3.4. Instead of putting an extra shell command after a function definition, we put it at the beginning (see the following example). We then run Bash, which is vulnerable to the Shellshock attack. Will the shell command `echo world` be executed?

```
$ export foo='echo world; () { echo hello;}'
$ bash
```

3.5. For the Shellshock vulnerability to be exploitable, two conditions need to be satisfied. What are these two conditions?

3.6. How do user inputs get into a remote a CGI program (written in Bash) in the form of environment variables?

3.7. Instead of using a function definition in the Shellshock attack against CGI programs, can we directly put shell commands inside the `User-Agent` field, so when Bash is triggered, the shell command can be executed?

3.8. ★
Why was this mistake made? And what lesson did you learn from this mistake?

3.9. There is another way to send inputs to a CGI program. That is to attach the input in the URL. See the following example.

```
http://www.example.com/myprog.cgi?name=value
```

Can we put our malicious function definition in the value field of the above URL, so when this value gets into the CGI program `myprog.cgi`, the Shellshock vulnerability can be exploited?

3.10. We run "`nc -l 7070`" on Machine 1 (IP address is `10.0.2.6`, and we then type the following command on Machine 2. Describe what is going to happen?

```
$ /bin/cat < /dev/tcp/10.0.2.6/7070 >&0
```

3.11. Please describe how you would do the following: run the `/bin/cat` program on Machine 1; the program takes its input from Machine 2, and print out its output to Machine 3.

3.12. Consider the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

extern char **environ;

int main()
{
  char *args[] =
  {
    "/bin/sh", "-c",
    "/bin/ls", NULL
  };
  pid_t pid = fork();

  if(pid == 0) {
     /* child */
     printf("child\n");
     execve(args[0], &args[0], NULL); ①
  }
  else if(pid > 0) {
     /* parent */
     printf("parent\n");
  }
  return 0;
}
```

The program forks a child process, and executes the `/bin/ls` program using `/bin/sh`, which is a symbolic link to `/bin/bash`. The program is executed as the following. Explain what the output of the program will be and why.

```
$ gcc prog.c -o prog
$ export foo='() { echo hello; }; echo world;'
$ ./prog
```

3.13. Let's make a change to the code in Problem 3.12.. We change the code in Line ① to the following. Please redo Problem 3.12. with this change made.

```
     execve(args[0], &args[0], environ);
```

3.14. Consider a PHP program running as Apache module, and a CGI program.

```
-----------------------------------
The PHP program (test.php):

<?php
   system("/bin/ls -l")
?>

-----------------------------------
The CGI program (test.cgi):

#!/bin/sh
/bin/ls -l
```

Both programs invoke `/bin/ls` command in a new shell process (`/bin/sh` points to `/bin/bash`). If the programs are invoked as the following, please explain the difference in effect of the Shellshock vulnerability on these two cases. What conditions are necessary to exploit shellshock in either case?

```
$ curl -A "() { echo hello; }; echo world;"
    http://localhost/test.php
$ curl -A "() { echo hello; }; echo world;"
    http://localhost/test.cgi
```