Secure System Design: Threats and Countermeasures
CS392

Date: 25th Jan 2019 Assignment 2
Submission Filename: assign2.pdf Due Date: 30th Jan 2019
Full Marks 40

Description

The learning objective of this assignment is for students to understand how environment variables affect program and system behaviors. Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They are used by most operating systems, since they were introduced to Unix in 1979. Although environment variables affect program behaviors, how they achieve that is not well understood by many programmers. As a result, if a program uses environment variables, but the programmer does not know that they are used, the program may have vulnerabilities. In this assignment, students will understand how environment variables work, how they are propagated from parent process to child, and how they affect system/program behaviors. We are particularly interested in how environment variables affect the behavior of `Set-UID` programs, which are usually privileged programs.

Also, we would like to exploit the shellshock vulnerability which was identified on September 24, 2014 in Bash. Nicknamed Shellshock, this vulnerability can exploit many systems and be launched either remotely or from a local machine. In one of the tasks of this assignment, students need to work on this attack, so that they can understand the Shellshock vulnerability.

Task 1

In this task, you need to study how a child process gets its environment variables from its parent. In Unix, *fork()* creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child (you may refer the manual of fork() by typing the following command: *man fork*). In this task, you need to check whether the parents environment variables are inherited by the child process or not.

1. Initially check whether */bin/sh* is pointing to */bin/dash* or not. If not then point it to */bin/dash*.
2. Please compile and run the following program, and describe your observation. Because the output contains many strings, you should save the output into a file, such as using *a.out* > *childFilename* (assuming that *a.out* is your executable file name).

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

extern char **environ;

void printenv()
{
  int i = 0;
  while (environ[i] != NULL) {
   printf("%s\n", environ[i]);
   i++;
  }
}
void main()
{
  pid_t childPid;

  switch(childPid = fork()) {
   case 0: /* child process */
   printenv(); /* Line (A) */
```

```
      exit(0);
     default: /* parent process */
     //printenv(); /* Line (B) */
     exit(0);
    }
   }
```

3. Now comment out the *printenv()* statement in the child process case (Line A), and uncomment the *printenv()* statement in the parent process case (Line B). Compile and run the code again, and describe your observation. Save the output in another file.
4. Compare the difference of these two files using the *diff* command. Please draw your conclusion.

10 Marks

Task 2

Because of the shell program invoked, calling system() within a *Set-UID* program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as *PATH*; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the *Set-UID* program. In Bash, you can change the *PATH* environment variable in the following way (this example adds the directory */home/seed* to the beginning of the *PATH* environment variable):

```
$ export PATH=/home/seed:$PATH
```

The *Set-UID* program below is supposed to execute the */bin/ls* command; however, the programmer only uses the relative path for the *ls* command, rather than the absolute path:

```
int main()
{
  system("ls");
  return 0;
}
```

Please compile the above program, and change its owner to root, and make it a Set-UID program. Can you let this *Set-UID* program run your code instead of */bin/ls*? If you can, is your code running with the root privilege? Describe and explain your observations. 10 Marks

Task 3

Now, link */bin/sh* to *zsh* and repeat the above attack. Check whether the attack succeeds or not?

```
$ sudo rm /bin/sh
$ sudo ln -s /bin/zsh /bin/sh
```

10 Marks

Task 4

In this task, you need to launch a Shellshock attack on a remote web server. Many web servers enable CGI, which is a standard method used to generate dynamic content on Web pages and Web applications. Many CGI programs are written using shell scripts. Therefore, before a CGI program is executed, a shell program will be invoked first, and such an invocation is triggered by a user from a remote computer. If the shell program is a vulnerable Bash program, we can exploit the Shellshock vulnerable to gain privileges on the server. In this task, you need to set up a very simple CGI program (called *myprog.cgi*) like the following. It simply prints out "Hello World" using a shell script.

```
#!/bin/bash_shellshock     (A)
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

Please make sure you use */bin/bash_shellshock* in Line (A), instead of using */bin/bash*. The line specifies what shell program should be invoked to run the script. We do need to use the vulnerable Bash in this task. Please place the above CGI program in the */usr/lib/cgi-bin* directory and set its permission to 755 (so it is executable). You need to use the root privilege to do these, as the folder is only writable by the root. This folder is the default CGI directory for the Apache web server.

To access this CGI program from the Web, you can either use a browser by typing the following URL: *http://localhost/cgi-bin/myprog.cgi*, or use the following command line program *curl* to do the same thing:

```
$ curl http://localhost/cgi-bin/myprog.cgi
```

In our setup, we run the Web server and the attack from the same computer, and that is why we use *localhost*. In real attacks, the server is running on a remote machine, and instead of using *localhost*, we use the *hostname* or the IP address of the server.

Passing Data to Bash via Environment Variable To exploit a *Shellshock* vulnerability in a Bash-based CGI program, attackers need to pass their data to the vulnerable Bash program, and the data need to be passed via an environment variable. In this task, we need to see how we can achieve this goal. You can use the following CGI program to demonstrate that you can send out an arbitrary string to the CGI program, and the string will show up in the content of one of the environment variables.

```
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ            //Line (A)
```

In the code above, Line (A) prints out the contents of all the environment variables in the current process. If your experiment is successful, you should be able to see your data string in the page that you get back from the server. In your report, please explain how the data from a remote user can get into those environment variables.

After the above CGI program is set up, we can now launch the *Shellshock* attack. The attack does not depend on what is in the CGI program, as it targets the Bash program, which is invoked first, before the CGI script is executed. Your goal is to launch the attack through the URL *http://localhost/cgi-bin/myprog.cgi*, such that you can achieve something that you cannot do as a remote user. In this task, you should demonstrate the following:

- Using the *Shellshock* attack, is it possible to steal the content of a secret file from the server?
- Will you be able to steal the content of the shadow file */etc/shadow*? Why or why not?

10 Marks

Submission

You need to submit a detailed report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are *interesting* or *surprising*. Add necessary snapshots of your experiment wherever applicable in support of your observation. Upload your file using following link only.

http://172.16.1.252/~samrat/CS392/submission/