Reading Course Report:
Demo of exploitation of Shell-shock vulnerability

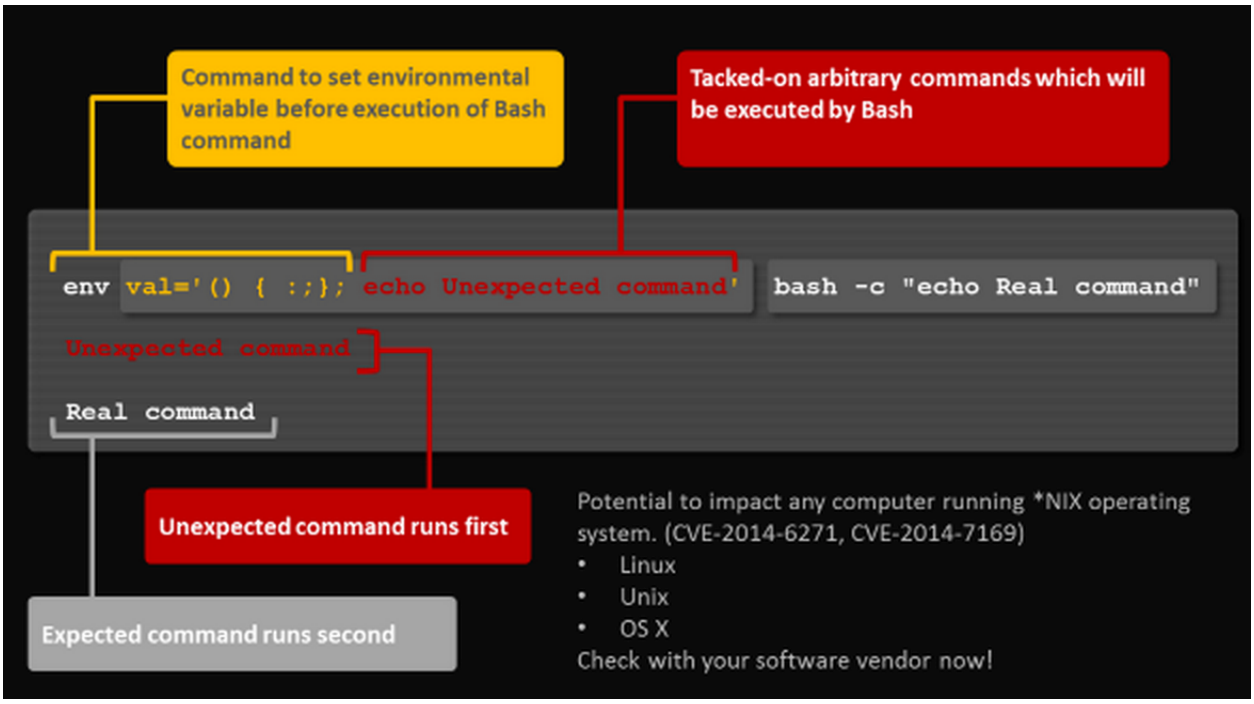
Yuan Jiang
F00227C

Introduction

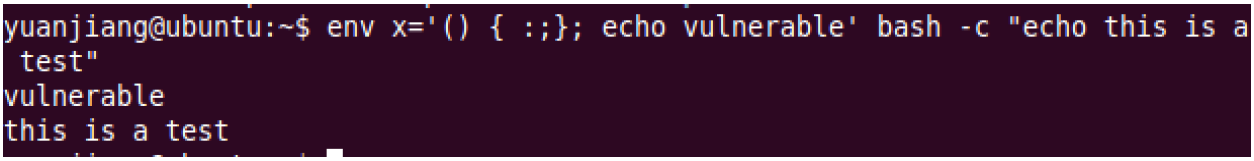
The bug known as “shell-shock” or “bash bug” is discovered in the mid-2014. It has CVE number CVE-2014-6271. It is not yet been fully patched. The vulnerability is associated with Bash, which is an essential component in most Linux (such as Ubuntu) or Unix (such as Mac OS) systems. I have tested Ubuntu 10.04 using the testing code in the follow sections, and confirmed that it is vulnerable to this bug.

The attacker can exploit this vulnerability to insert malicious code into the environment variables to be sent to Bash. The server would then wrongly execute those malicious code embedded in environment variables, thus giving attacker root, or certain level of control over the victim web server.

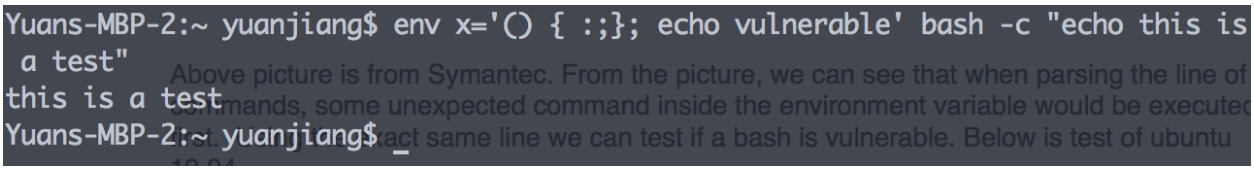
Detailed explanation and test of vulnerability



Above picture is from Symantec. From the picture, we can see that when parsing the line of commands, some unexpected command inside the environment variable would be executed first. Using the exact same line we can test if a bash is vulnerable. Below is test of ubuntu 10.04.



The newest Mac OS X Yosemite seems to fix this bug.



Exploit Demo

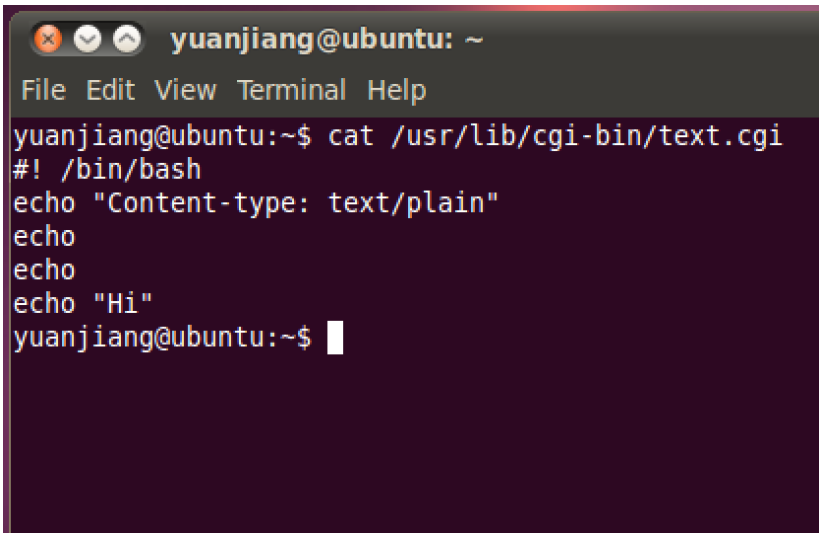
The exploit references the example described in stack exchange[1].

In order to run Ubuntu 10.04, I have downloaded the ISO file and also used VM fusion 7 for Mac OS. At first attempt I used virtual box to run the OS, but later found out I cannot ping from my Mac to the Virtual Machine. It seems many people are having this kind of issue online.

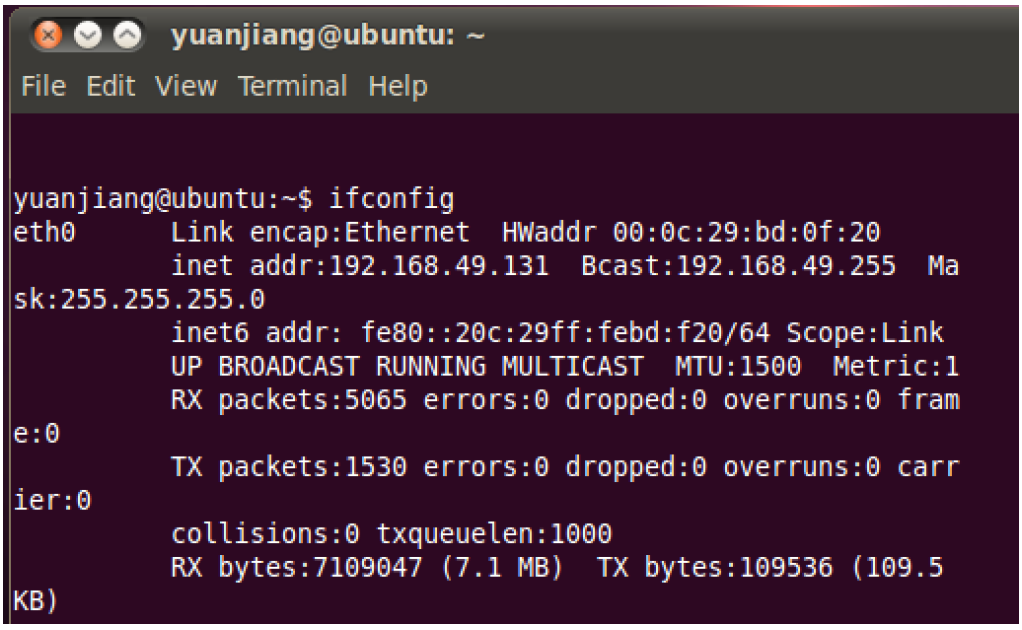
Upon installing Ubuntu 10.04 on VM fusion, apache2 sever are installed and the service is started first accordingly. we don't have to configure the host and default settings are sufficient enough.

The next step is to create a text.cgi file under **/usr/lib/cgi-bin/text.cgi.** The content of this file is shown below:

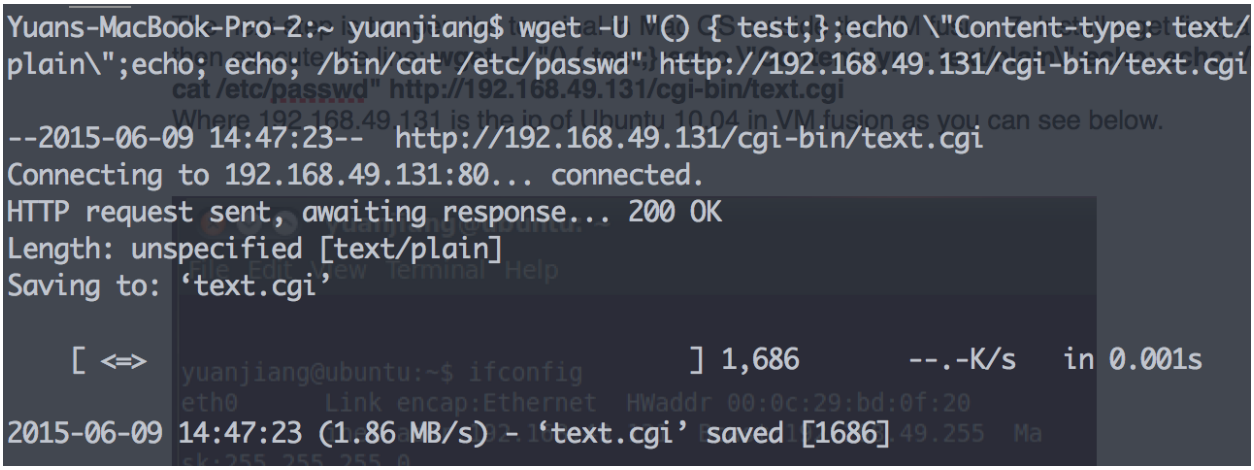
[1] http://security.stackexchange.com/questions/68122/what-is-a-specific-example-of-how-the-shellshock-bash-bug-could-be-exploited



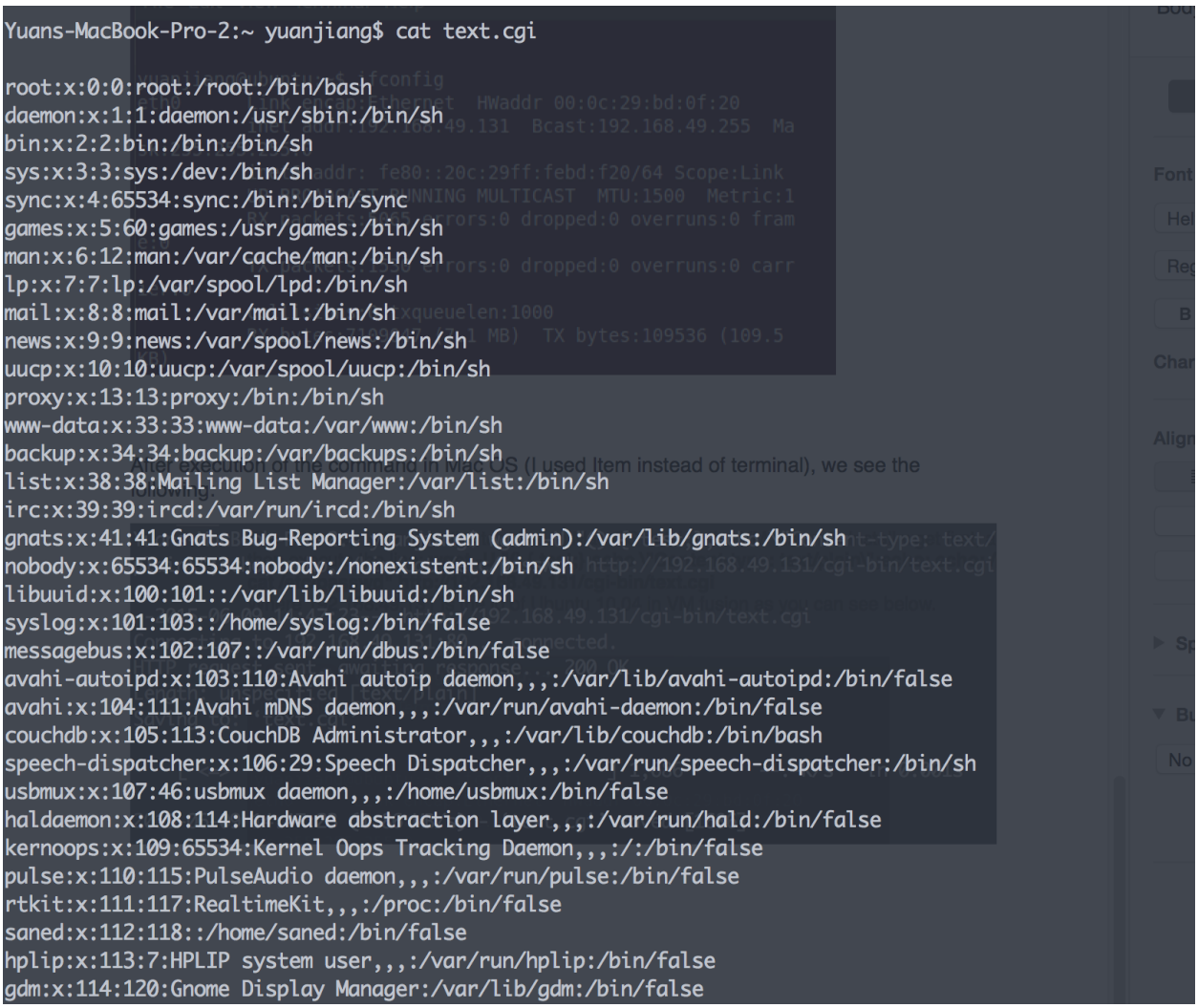
The next step is to open the terminal in Mac OS outside the VM fusion 7. Install wget first, and then execute the line: **wget -U "() { test;};echo \"Content-type: text/plain\";echo; echo; /bin/cat /etc/passwd" http://192.168.49.131/cgi-bin/text.cgi**
Where 192.168.49.131 is the ip of Ubuntu 10.04 in VM fusion as you can see below.



After execution of the command in Mac OS (I used Item instead of terminal), we see the following:



cat cgi we will get all the content of /ect/passwd in the victim web server. shown as below



The exploit demonstrate that the attacker can retrieve the file **/ect/passwd** by passing malicious code (in this case as simple as **/bin/cat /etc/passwd**), theoretically we can run lots lots of instructions replacing **/bin/cat /etc/passwd** and hopefully to get it executed in the web server. For example another possible exploit is to use **sleep** to initiate **a DOS attack.**

Path the source code

The erroneous piece of code is in the importing of the function variables in variables.c:

```
/* Initialize the shell variables from the current environment.
   If PRIVMODE is nonzero, don't import functions from ENV or
   parse $SHELLOPTS. */
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 && STREQN ("() {", string, 4))
      {
        [...]
        parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
        [...]
      }
}
```

```
/* Parse and execute the commands in STRING.  Returns whatever
   execute_command () returns.  This frees STRING.  FLAGS is a
   flags word; look in common.h for the possible values.  Actions
   are:
    (flags & SEVAL_NONINT) -> interactive = 0;
    (flags & SEVAL_INTERACT) -> interactive = 1;
    (flags & SEVAL_NOHIST) -> call bash_history_disable ()
    (flags & SEVAL_NOFREE) -> don't free STRING when finished
    (flags & SEVAL_RESETLINE) -> reset line_number to 1
*/
int
parse_and_execute (string, from_file, flags)
     char *string;
     const char *from_file;
     int flags;
{
```

So everything that's passed to the function gets executed as if it would be an ordinary bash command. The patch introduces flags **SEVAL_FUNCDEF** and **SEVAL_ONECMD** that can be passed in the flags field to parse_and_execute, the patch also adds functionality to **parse_and_execute** to comply with those new flags, and changes the call to **parse_and_execute** to pass those flags[2]:

```
+ #define SEVAL_FUNCDEF 0x080       /* only allow function definitions */
+ #define SEVAL_ONECMD  0x100       /* only allow a single command */
-    parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
+     /* 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);
```

Aside form the detailed source code of the patch, from a user's perspective all we have to do is first path the current version of bash we have, and then run the testing code above to verify that the updated bash is error free.

---

[2] http://security.stackexchange.com/questions/68448/where-is-bash-shellshock-vulnerability-in-source-code