Radware Security Research
Reverse Engineering a Sophisticated DDoS Attack Bot
Author: Zeev Ravid
July 2015
Introduction

In July 2015, Radware’s Emergency Response Team (ERT) noticed a significant increased usage of the Tsunami SYN Flood attack against a large customer.

During the investigation of these attacks, Radware Security Researchers obtained a sample of a malware binary which was used to generate these DDoS attacks.

Radware’s ERT deployed the malware in an isolated and controlled environment to study its behavior and its different attack vectors.

During a period of 10 days (June 14-23, 2015), Radware’s ERT monitored greater than 2000 attacks against more than 60 different targets across 7 different countries.

This report presents an analysis of the attack, the sophisticated attack tool and an inside look into a Tsunami DDoS attack.
**DDoS-as-a-Service**

Upon analysis, the attack patterns of this DDoS bot indicates that the attacks were likely initiated through a professional malware operator who provides “DDoS for hire”\(^1\) services, selling its DDoS capabilities to random bidders.

Even if one lacks the relevant expertise, it’s easy and inexpensive to launch an attack.

There are increasingly large numbers of “shady” websites providing easy-to-use web interfaces that can be used to launch DDoS attacks. Dozens of websites, referring to themselves as “booter” or “stresser,” openly advertise themselves on the Internet and accept payments with Bitcoin or PayPal, some starting as low as $9.99 per month.

Below are sample screenshots:

\(^1\) [http://krebsonsecurity.com/category/ddos-for-hire/]
To launch attacks with a DDoS for hire service, registered users gain access to the admin panel and additional features like attack history, network statistics, and a ticket system for opening customer support incidents.

Many of those services have more than 20,000 registered users:

Panel 1

Panel 2
Most of the panels also show global system statistics for nearly 5,000 DDoS attacks per week:

![Panel 1](image1)

![Panel 2](image2)

The charts in Panel 1 and Panel 2 above show that these systems are truly capable of delivering a 10+ Gbps attack, which could become a significant threat to a critical business system.

**About the Tsunami SYN Flood Attack**

The Tsunami SYN Flood involves sending a massive amount of TCP SYN packets with a non-zero payload, unlike the common SYN packets which don’t contain data.

Unfortunately, it is easy for an attacker to create SYN packets and use them to launch a volumetric DDoS attack. This attack can saturate the Internet connection of your organization even before hitting any stateful network device like the Firewall and Load Balancer.

More detailed information about the Tsunami SYN attack can be found [here](#).
Inside Look at a DDoS Attack

The image below shows network traffic captured during the DDoS attack. Notice the SYN packets with the Len=896 bytes payload highlighted in red. As one can see, the attacker was sending a high rate of SYN packets.

<table>
<thead>
<tr>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.743719</td>
<td>52.17.122.123</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743722</td>
<td>61.175.73.197</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743725</td>
<td>128.209.2.242</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743727</td>
<td>220.178.135.396</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743730</td>
<td>52.17.122.123</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743733</td>
<td>52.11.236.162</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743735</td>
<td>52.74.83.156</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743737</td>
<td>128.209.2.242</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743738</td>
<td>216.12.100.142</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743739</td>
<td>211.142.58.135</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743822</td>
<td>128.209.2.242</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743833</td>
<td>52.11.236.162</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743835</td>
<td>58.49.59.192</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743846</td>
<td>52.17.122.123</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743847</td>
<td>52.17.122.123</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743848</td>
<td>128.209.2.242</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743849</td>
<td>52.17.122.123</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
<tr>
<td>0.743850</td>
<td>52.17.122.123</td>
<td>202.168.183.242</td>
<td>TCP</td>
<td>896</td>
<td>Seq=0 Win=65535 Len=896</td>
</tr>
</tbody>
</table>

The table on the following page shows a list of the top attackers’ IP addresses and the number of SYN packets they’ve sent over a period of about 70 seconds.
The IPs highlighted in yellow appear to be real servers with real IPs, and responsible for generating over 80% of the traffic. Most of the real IPs are hosted on the Amazon EC2 network (52.*.*.*). The rest of the traffic was spoofed, and used more than 50,000 random IPs belonging to about six different large network segments.

This investigation suggests that the spoofed traffic was generated by 10-15 compromised or rented servers that were all running on hosting providers which do not prevent their clients from sending spoofed IP traffic.
The next section will provide analysis of the tool that has been used for generating these attacks.

**Tracking Down the Attackers**

Radware’s ERT deployed the malware in an isolated and controlled environment to study its behavior and attack vectors.

Upon close examination, it was discovered that the bot was communicating with an active C&C (Command & Control) server, which had been instructing it to launch repeated attacks against different targets. The duration of each attack was between several minutes to several hours.

During a period of 10 days (June 14-23, 2015), we monitored greater than 2000 attacks against more than 60 different targets. The duration of each attack was between several minutes to several hours.

The following list of C&C servers is hardcoded in the binary:

1. 66.102.253.30
2. zhegege.3322.org
3. 122.224.48.63

The targets of the attack are mostly servers located in China:

<table>
<thead>
<tr>
<th># Targets</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>China</td>
</tr>
<tr>
<td>4</td>
<td>Philippines</td>
</tr>
<tr>
<td>5</td>
<td>Hong Kong</td>
</tr>
<tr>
<td>2</td>
<td>United States</td>
</tr>
<tr>
<td>1</td>
<td>Thailand</td>
</tr>
<tr>
<td>1</td>
<td>Malaysia</td>
</tr>
<tr>
<td>1</td>
<td>Japan</td>
</tr>
</tbody>
</table>
Attacked targets include:

<table>
<thead>
<tr>
<th>IP</th>
<th>CC</th>
<th>Country</th>
<th>Network</th>
<th>Suffix</th>
<th>ASNum</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>208.89</td>
<td>US</td>
<td>United States</td>
<td>208.89::</td>
<td>22</td>
<td>AS111</td>
<td>Li.</td>
</tr>
<tr>
<td>120.52</td>
<td>CN</td>
<td>China</td>
<td>120.52::</td>
<td>16</td>
<td>AS131</td>
<td>C.</td>
</tr>
<tr>
<td>43.225</td>
<td>MY</td>
<td>Malaysia</td>
<td>43.225::</td>
<td>22</td>
<td>AS131</td>
<td>C.</td>
</tr>
<tr>
<td>119.28</td>
<td>CN</td>
<td>China</td>
<td>119.28::</td>
<td>19</td>
<td>AS131</td>
<td>C.</td>
</tr>
<tr>
<td>117.27</td>
<td>CN</td>
<td>China</td>
<td>117.27::</td>
<td>22</td>
<td>AS131</td>
<td>Fast</td>
</tr>
<tr>
<td>112.10</td>
<td>CN</td>
<td>China</td>
<td>112.10::</td>
<td>15</td>
<td>AS179</td>
<td>ad.</td>
</tr>
<tr>
<td>202.16</td>
<td>PH</td>
<td>Philippines</td>
<td>202.16::</td>
<td>23</td>
<td>AS188</td>
<td>G.</td>
</tr>
</tbody>
</table>
| 45.64.6 | HK | Hong Kong    | 45.64.6:: | 24     | AS199 | Inc.
| 61.147 | CN | China         | 61.147:: | 23     | AS233 | A.  |
| 161.20 | JP | Japan         | 161.20:: | 18     | AS365 | S.  |
| 120.55 | CN | China         | 120.55:: | 16     | AS377 | H.  |
| 210.56 | HK | Hong Kong     | 210.56:: | 20     | AS383 | S.  |
| 182.14 | CN | China         | 182.14:: | 18     | AS383 | C.  |
| 45.34.6 | US | United States | 45.34.6:: | 16     | AS402 | P.  |
| 122.22 | CN | China         | 122.22:: | 14     | AS419 | C.  |
| 113.21 | PH | Philippines   | 113.21:: | 23     | AS455 | M.  |
| 122.19 | CN | China         | 122.19:: | 14     | AS484 | C.  |
| 103.26 | TH | Thailand      | 103.26:: | 22     | AS556 | 60. |
| 14.29.3 | CN | China         | 14.29.3:: | 19     | AS583 | G.  |
| 118.19 | CN | China         | 118.19:: | 18     | AS589 | S.  |
| 43.227 | CN | China         | 43.227:: | 19     | AS596 | N.  |
| 120.20 | CN | China         | 120.20:: | 16     | AS983 | G.  |

Every day different servers are attacked on different ports:

<table>
<thead>
<tr>
<th>Count</th>
<th>Date</th>
<th>IP</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6/25/2015</td>
<td>116.31</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>6/25/2015</td>
<td>117.27.2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>6/25/2015</td>
<td>117.27</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>6/25/2015</td>
<td>117.27</td>
<td>80</td>
</tr>
<tr>
<td>54</td>
<td>6/25/2015</td>
<td>118.193.1</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>6/25/2015</td>
<td>118.193</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>6/25/2015</td>
<td>122.193</td>
<td>80</td>
</tr>
<tr>
<td>12</td>
<td>6/25/2015</td>
<td>122.228.2</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>6/25/2015</td>
<td>122.228.2</td>
<td>9800</td>
</tr>
<tr>
<td>31</td>
<td>6/25/2015</td>
<td>14.29</td>
<td>80</td>
</tr>
<tr>
<td>81</td>
<td>6/25/2015</td>
<td>14.29</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>6/25/2015</td>
<td>203.202</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>6/25/2015</td>
<td>218.90</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>6/25/2015</td>
<td>42.19</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>6/25/2015</td>
<td>42.19</td>
<td>80</td>
</tr>
<tr>
<td>26</td>
<td>6/25/2015</td>
<td>42.19</td>
<td>80</td>
</tr>
<tr>
<td>15</td>
<td>6/25/2015</td>
<td>43.227</td>
<td>1520</td>
</tr>
<tr>
<td>8</td>
<td>6/25/2015</td>
<td>43.227</td>
<td>22</td>
</tr>
<tr>
<td>34</td>
<td>6/25/2015</td>
<td>61.154.1</td>
<td>9800</td>
</tr>
</tbody>
</table>
Analyzing the Command Packets Between the Bot and C&C

The bot periodically sends information about itself to the C&C server. The data is “encrypted” with a fixed 16 bytes XOR key (BB2FA36AAA9541F0).

This is what a sample packet looks like – highlighted in red is the current C&C server address:

After decryption, this is the data that is being sent:
It includes information about the current system:

- Linux 2.6.18-194.el5,
- 2x2000 MHz CPUs
- 32 bit i686 processor

After few seconds, the C&C server sends a special packet instructing the bot to **attack**. This would be a 304 bytes TCP packet that looks as follows:
After decryption of the payload data:

**Green** – the bot external IP address as seen by the C&C server

**Red** – the target IP address 0xb783XXXX (**183.131.X.X**) on port 0x0050 (port 80)

**Blue** – attack method – 05 means SYN flood. 0x03e7 means packet size – 999 bytes.

Although the bot is capable of launching different attack methods, in this report the bot was only using the “05” method – e.g. Tsunami SYN flood, with different packet sizes.

Shortly after receiving this message, the bot will start attacking his target by sending large SYN packets:
Network Isolation

The previous screenshot shows a very slow attack, running at 5 packets per second. The image above was produced in a controlled sandbox environment, developed by Radware’s ERT for researching and monitoring malware activity without the risk of participation in any attack activity.

Without the limited sandbox environment, the malware would have sent about 100,000 packets per second, utilizing the entire 1Gbps virtual network port. The screenshot below shows the OS statistics on VM the malware was running on – the 105 MB/sec outgoing traffic is highlighted in red:

DDoS Bot Performs Self-Diagnostic Tests

The malware, together with the C&C, is engaging in sophisticated ‘self-tests’ to both check its ability to send spoofed IP traffic and perform a ‘self-benchmark’ test to checking its flooding ability.
In the image below, one can see the malware running on 10.0.5.105, communicating with Google’s DNS server (8.8.8.8) to resolve the C&C domain (zhegege.3322.org), which results in the IP 122.224.51.128. Then it begins to send UDP packets from different ‘spoofed’ IPs, targeting the C&C server itself.

In another screenshot, one can see the malware generating traffic from additional random IPs, again targeting the C&C server:

This behavior shows how the malware manages to “self-discover” whether it is capable of sending spoofed IP traffic, and if so, from which subnets. Later, the malware uses this information to perform a large DDoS attack from thousands of spoofed IP addresses, which are very similar to the 50,000 Unique IPs seen in the attack investigated previously.
Reverse Engineering of the Malware Sample

This malware belongs to the **Linux XOR/DDoS** family, which has been active throughout the past year and has been previously analyzed by Avast\(^2\), Fireeye\(^3\), and others\(^4\). During the writing of this paper, additional infections have been reported\(^5\)\(^6\).

It is a sophisticated bot, written in C or C++, and remotely controlled and operated by a central server. It has the following characteristics:

- Uses an XOR key for encryption/obfuscation
- Uses a persistence mechanism, which makes it difficult for an average user to remove it from an infected Linux machine
- Periodically connects to a list of C&C (Command and Control) servers and receives commands such as:
  - Start attack
  - Stop attack
  - Download a file and execute, which is a remote upgrade mechanism
- Supports multiple attack methods – SYN, ACK, DNS
- Supports HTTP GET/POST commands
- Supports sending DNS Queries
- Supports IP spoofing

\(^2\) [https://www.fireeye.com/blog/threat-research/2015/02/anatomy_of_a_brute.html](https://www.fireeye.com/blog/threat-research/2015/02/anatomy_of_a_brute.html)
\(^4\) [https://isc.sans.edu/diary/XOR+DDOS+Mitigation+and+Analysis/19827](https://isc.sans.edu/diary/XOR+DDOS+Mitigation+and+Analysis/19827)
A sample of the binary can be found on VirusTotal with the following SHA256 hash:

498f3348df1b6804db2692e4f937d7cbefd71916e83a9421347077fb1cdafa95

The malware was first submitted to VirusTotal on May 21, 2015, approximately 1 month prior to the writing of this paper. As of May 26, the antivirus detection ratio is only 18/57.
Malware Static Analysis

The binary is a Linux ELF 32 bit executable, built for the 32 bit x86 Intel platform.

It was compiled and statically linked for an old Linux kernel version – 2.6.9.

It is also an unstripped binary, meaning it still contains Debug Symbols with visible source code file names:
Function names are in the code section, and variable names are in the data section:

Below is a diagram of the execution flow of important functions in this malware:
The binary contains the following strings:

103.25.9.228  
103.25.9.229  
8.8.8.8  
/etc/cron.hourly/udev.sh 
/etc/init.d/%s 
/etc/ld.so.cache 
/etc/localtime 
/etc/rc%d.d/S90%s 
/etc/resolv.conf 
/etc/rc.d/rc%d.d/S90%s 
/etc/udev 
/etc/udev.d/90udev.conf.d 
/etc/suid -debug 
/etc/cron.hourly/udev.sh 
/lib/libgcc4.4.4.so 
/proc/%d/exe 
/proc/self/exe 
glibc 2.5 
GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-46) 
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin 
GET %s HTTP/1.1 
POST %s HTTP/1.1 
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; TencentTraveler; .NET CLR 1.1.4322) 
cp /lib/libgcc4.so /lib/libgcc4.4.so 
find /proc/meminfo 
find /proc/cpuinfo 
find /proc 
find /lib/libgcc4.so 

Using a simple XOR obfuscation, it contains additional “hidden” strings. These include:

1. 66.102.253.30:2866|zhegege.3322.org:2866|122.224.48.63:2866 
3. /var/run/udev.pid 
4. /var/run 
5. /lib/libgcc4.so 

The first string is a list of C&C servers to which the malware will try to connect every few seconds.

The second string is a URL which will be fetched after the bot is running for 72 hours. The complete URL will look as follows:

The malware will attempt to download a file from that URL, and if successful – it will execute it.

Another group of obfuscated strings is a list of “fake” process names which the malware is using during runtime for hiding its presence. When a user will use the “ps aux” command, for example, he will not see the process name of the malware, but rather one of the following fake strings:

cat resolv.conf
sh
bash
su
ps -ef
ls
ls -la
top
netstat -an
netstat -antop
grep "A"
sleep 1
cd /etc
echo "find"
ifconfig eth0
ifconfig

**Binary Replication**

During the malware runtime, the process will create multiple threads. One of these threads is the daemon thread, which is responsible for ensuring both the process will be executed automatically after reboot and that it is difficult to terminate the process. This thread will keep copying the original file to a new, randomly generated 10-character file name. If the original process is killed, then the new one will automatically take over execution.

Additionally, the newly created binary contains an extra 11 bytes which are appended to the end of the file. View the below image showing the differences between the original file (mtjrnexwtq) and the one that was executed automatically (uubtuadsrx) after we manually killed the first one:
Process Name Hiding

As mentioned in the previous section, the malware will attempt to hide its process name by changing to a fake process name during runtime. In the screenshot below one can see the original malware process is highlighted in red with a fake process name “id”, while the original name is actually “gupccjvpff”.

Notice the “532:05” text in the CPU column which indicates this process has been keeping the CPU very busy attacking targets. One can see a group of five processes highlighted in blue with fake names. These are processes through which the malware continuously creates and replicates itself to new binaries with new names, before killing and deleting the previous one.
If the malware is run using `strace`, it creates a copy of itself by copying `/lib/libgcc4.so` to a new random file `/usr/bin/jyyddrbtp`:

```
open("/lib/libgcc4.so", O_RDONLY) = 4
lseek(4, 0, SEEK_SET)
open("/usr/bin/jyyddrbtp", O_RDONLY|O_CREAT, 0777) = 5
lseek(5, 0, SEEK_SET)
read(5, 
write(5, 
close(4)
close(5)
gettimeofday({1435885201, 112089}, NULL) = 0
write(5, 
clone(child_stack=0, flags=CLONE_CHILD_CLEARDIR|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xbdc3888) = 23601
waitpid(23601, 0, 0) = -1 ECHILD (No child processes)
open("/usr/bin/jyyddrbtp", O_RDONLY) = -1 EEXIST (File already exists)
gettimeofday({1435885201, 113663}, NULL) = 0
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
close(4)
close(5)
Summary

Malware authors appear to be relentlessly writing new malware for fun and profit.

Security vulnerabilities in servers and applications are discovered daily. Unfortunately, malware authors exploit those vulnerabilities for the purpose of gaining access to remote systems, infecting them, and then assembling an army of “zombies” to launch DDoS attacks, send spam, and more.

Even worse, malware is rapidly infecting servers, personal routers, and other network equipment for which the owner either did not change the default password, or simply used a trivial one. Weak passwords make it easy to launch brute force attacks.

Defending against DDoS attacks can be quite difficult. Unfortunately, there are still many ISPs and hosting companies that do not prevent IP Spoofing by filtering their ingress traffic and fail to implement (BCP38 / RFC 2827). Because IP blacklisting is not possible, it’s even more difficult to defend against DDoS attacks.

Over the last year there has seen a significant increase in the adoption and use of Linux-based malware to infect SOHO routers, PBX systems, and various Linux servers. This malware presents many significant security threats with unique characteristics. Notably, a known vulnerability and target for exploitation are the many powerful Linux-based systems which do not enforce password policies. They are often left overlooked, and instead, become a sitting target for DDoS malware infection.

7 https://www.incapsula.com/blog/ddos-botnet-soho-router.html
**Recommended Steps**

DDoS protection mechanisms should prevent both volumetric and sophisticated attacks at the same time. It is imperative for business critical applications to be able to accurately mitigate attacking traffic while allowing uninterrupted, legitimate traffic to flow through the network.

The following best practices are recommended:

1. Enforce a strong password security policy for servers, workstations and network appliances.
2. Ensure systems are up-to-date and patched with all critical security fixes.
3. Use Antivirus for both Windows AND Linux systems.
4. Check router ACLs and firewall rules, and implement BCP38.
5. Perform a security audit at least once or twice a year.

Whether currently experiencing a DDoS attack, or concerned about DDoS protection mechanisms and attack mitigation solutions for safeguarding business operations, please contact Radware’s Emergency Response Team (ERT) for information and assistance.

For additional research reports and DDoS prevention techniques, please visit Radware’s [DDoS Security Research Center](#).