AKAMAI THREAT ADVISORY

Kaiten/STD router DDoS Malware

Risk Factor: Medium

Akamai
FASTER FORWARD

Issue Date: 10/1/16

Threat Advisory: Kaiten/STD Router DDoS Malware

Editor's Note: The research published in this advisory predates the very public attacks on *www.krebsonsecurity.com* in mid-September. The malware examined in this advisory has since been implicated in those attacks. Akamai has chosen to publish this existing research without specific reference to those attacks as the mitigation and cleanup steps are important to publicize. We will be publishing further analysis that includes reference to more recent in the coming weeks.

1.0 / Overview /

Akamai SIRT is investigating a malware variant of Kaiten/STD specifically designed to target networking devices used in Small Office & Home Office (SOHO) as well as DVRs, IP Cameras and other IoT devices. This malware includes an extensive list of available attack vectors along with the ability to execute arbitrary commands and take full control over the infected system.

The malware is packed with a custom packer/encoder to hinder analysis. It's compiled for multiple architectures (MIPS, ARM, PowerPC, x86, x86_64) and uses a custom IRC-like communication protocol for C2 communications.

In our investigation we have seen the resulting botnets target companies operating in multiple business sectors. Although booter/stresser supported attacks have been more prevalent lately, the largest attacks had previously come from malware-based botnets like XOR and billgates. This variant of malware continues to expand on botnet-based attacks but is also being used in DDoS-For-Hire and extortion campaigns.

2.0 / Indicators of Binary Infection / The infection begins with either an exploitation of a vulnerability or common/guessed/default login credentials. Once the attacker gains access to the target system, a set of predefined commands are run to download and execute ultimately resulting in a malware infection. The malware in turn executes a long list of commands aimed at disabling firewall rules and gain persistence after reboot as shown in Figure 1.

```
echo “nameserver 8.8.8.8” > /etc/resolv.conf &
killall -9 telnetd > /dev/null 2>&1 &
service httpd stop > /dev/null 2>&1 &
service telnetd stop > /dev/null 2>&1 &
service sshd stop > /dev/null 2>&1 &
killall -9 utelnetd > /dev/null 2>&1 &
killall -9 dropbear > /dev/null 2>&1 &
killall -9 sshd > /dev/null 2>&1 &
killall -9 minihttpd > /dev/null 2>&1 &
kill -9 `cat /var/run/thttpd.pid` > /dev/null 2>&1 &
nvram set httpd_enable=0 > /dev/null 2>&1
nvram set http_enable=0 > /dev/null 2>&1
killall -9 httpd > /dev/null 2>&1 &
kill -9 `cat /var/run/httpd.pid` > /dev/null 2>&1 &
rm -rf /var/run/wgsh > /dev/null 2>&1 &
rm -rf /var/run/bbsh > /dev/null 2>&1 &
rm -rf /var/run/tt* > /dev/null 2>&1 &
rm -rf /tmp/tt* > /dev/null 2>&1 &
killall -9 arm > /dev/null 2>&1 &
killall -9 mips > /dev/null 2>&1 &
killall -9 mipsel > /dev/null 2>&1 &
killall -9 powerpc > /dev/null 2>&1 &
killall -9 ppc > /dev/null 2>&1 &
killall -9 daemon.armv4l.mod > /dev/null 2>&1 &
killall -9 daemon.i686.mod > /dev/null 2>&1 &
killall -9 daemon.mips.mod > /dev/null 2>&1 &
killall -9 daemon.mipsel.mod > /dev/null 2>&1 &
rm -rf /tmp/.xs/* > /dev/null 2>&1 &
iptables -A INPUT -p tcp --dport 22 -j DROP > /dev/null 2>&1 &
iptables -A INPUT -p tcp --dport 23 -j DROP > /dev/null 2>&1 &
iptables -A INPUT -p tcp --dport 80 -j DROP > /dev/null 2>&1 &
iptables -A INPUT -p tcp --dport 8080 -j DROP > /dev/null 2>&1 &
```

Figure 1: Full list of commands to disable firewall rules

Persistence is configured using the current user’s crontab configuration file. It’s using a single configuration command to start the malware every minute on the system. The crontab configuration shown in Figure 2.

```
* * * * * /{full path to malware}/{malware filename} > /dev/null 2>&1 &
```

Figure 2: Malware’s persistency configuration in crontab file

If we look at the malware’s header section, we can see that it’s statically compiled and initially includes only 2 sections. The first section with read-and-execute permissions acting as an unpacker stub routine and the second section with read-and-write permissions containing the packed and encoded data. We can use the readelf utility to display the malware’s Program Header as shown in Figure 3.

```
Program Headers:
 Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
 LOAD           0x000000 0x00c01000 0x00c01000 0x08828 0x08828 R E 0x1000
 LOAD           0x000448 0x0805f448 0x0805f448 0x00000 0x00000 RW  0x1000
```

Figure 3: Number of sections listed in the Program Header

The code in the packer stub routine includes a few syscalls to mmap and mprotect to create an additional section where the new unpacked code will be stored. To manually unpack this sample, we can set a breakpoint to catch all system calls, note the newly mapped memory page and set a hardware breakpoint on execute there. There are no anti-debugging tricks, so finding the Original Entry Point using the above method is straightforward. Once the Original Entry Point has been reached, it is possible to dump the unpacked instructions and continue analysis.

3.0 / Toolkit Analysis / The malware uses a predefined list of C2 IPs and a custom IRC protocol to connect and communicate with them. Once a connection is established to one of the predefined C2 IPs the infected host then authenticates with a dynamic password generated by the server and joins a private channel where it begins listening for commands.

The backdoor functionality of the malware allows the attacker to execute arbitrary commands on the infected machine. These commands include, but are not limited to:

- Download and execute additional malware/scripts
- Set the IP range for spoofed traffic
- Kill/View running processes
- Execute reverse shells
- Resolve internal network domain names
- Execute arbitrary shell commands with the permissions of the current user

The other major functionalities of the malware are aimed at launching DDoS Attacks. The malware presents a variety of attack/flood type options to the botnet operators — including, but not necessarily limited to:

- SYN Flood
- Spoofed/Non-spoofed UDP Flood with custom payloads
- Layer 7 HTTP GET/POST/HEAD Floods
- TCP Connection Floods
- XMAS Flood with customizable TCP Header Flags

4.0 / DDoS Attack Payloads / Many flood types can be generated using this malware. Figures 4 – 10 show the most common attack types and their respective characteristics.

The UDP Flood is generic but allows the control over the payload size and content by the operator as shown in Figures 4 and 5. It can be purposely crafted for bandwidth exhaustion attacks by setting a large payload size or alternatively can be aimed at resource exhaustion with no payload but a high packet-per-second threshold.

```
13:56:46.308717 IP 235.54.204.5.55520 > 127.0.0.1.80: UDP, bad length 24752 > 1472
13:56:46.308726 IP 91.247.24.42.47267 > 127.0.0.1.80: UDP, bad length 24752 > 1472
13:56:46.308736 IP 39.196.144.4.45652 > 127.0.0.1.80: UDP, bad length 24752 > 1472
13:56:46.308745 IP 63.176.1.37.36773 > 127.0.0.1.80: UDP, bad length 24752 > 1472
```

Figure 4: Spoofed UDP Flood with payload

```
13:57:40.348086 IP 127.0.0.1.55569 > 127.0.0.1.80: UDP, length 18
E...{.@.@.............P...-CONTROLLED_PAYLOAD
13:57:40.348097 IP 127.0.0.1.55569 > 127.0.0.1.80: UDP, length 18
E...{.@.@.............P...-CONTROLLED_PAYLOAD
13:57:40.348108 IP 127.0.0.1.55569 > 127.0.0.1.80: UDP, length 18
```

Figure 5: Non-spoofed UDP with controlled payload

The SYN Flood distinguishes itself by the length of the TCP Headers and their associated options. Figure 6 shows a 40-byte header containing the always present 20-byte TCP Header options.

```
0x0000:  4510 003c 0dbd 4000 4006 e97b 3924 8b4e  E..<..@.@..{9$.N
0x0010:  7f00 0001 87a7 0050 6aa2 b852 0000 0000  .......Pj..R....
0x0020:  a002 7d78 c787 0000 0204 05b4 0402 080a  ..}x............
0x0030:  0023 c183 5300 0000 0103 0300            .#..S.......
```

Figure 6: TCP Header outlining the different TCP Header characteristics

The following characteristics are always present at their current offsets:

- TCP Header size of 40 bytes including options
- Max Segment Size of 1460
- Selective Syn-Ack enabled
- NOP at offset 0x38
- Window Scale of x

```
13:56:05.928781 IP 158.78.19.118.15536 > 127.0.0.1.80: Flags [S], seq 709953562, win 32120,
options [mss 1460,sackOK,TS val 3395475 ecr 1291845632,nop,wscale 0], length 0
13:56:05.928793 IP 208.245.167.80.40945 > 127.0.0.1.80: Flags [S], seq 3928653631, win 32120,
options [mss 1460,sackOK,TS val 1998815 ecr 1023410176,nop,wscale 0], length 0
13:56:05.928799 IP 136.175.172.122.46989 > 127.0.0.1.80: Flags [S], seq 1587286634, win
32120, options [mss 1460,sackOK,TS val 7932474 ecr 1207959552,nop,wscale 0], length 0
13:56:05.928810 IP 143.93.204.102.18021 > 127.0.0.1.80: Flags [S], seq 1517504633, win 32120,
options [mss 1460,sackOK,TS val 2072700 ecr 956301312,nop,wscale 0], length 0
```

Figure 7: Spoofed SYN Flood

We have built a Berkeley Packet Filter in Figure 8 that can be used to match some of the traffic generated by the SYN Flood characteristics highlighted in Figure 7.

```
'((tcp[12] >> 4) << 2) == 40' and 'tcp[20:4] == 0x020405b4' and 'tcp[24:2] == 0x0402'
and 'tcp[36] == 1' and 'tcp[37:2] == 0x0303'
```

Figure 8: Barkeley Packet Filter to match traffic generated by SYN Flood attack vector

The TCP XMAS Flood can be set with attacker-controlled TCP flags which can either be run-of-the-mill standard combinations or more infrequently can contain rarely seen abnormal combinations. Sample attack traffic is shown in Figure 9 utilizing these abnormal flag combinations.

```
13:54:00.019647 IP 110.8.201.6.58740 > 127.0.0.1.80: Flags [FSRP.UEW], seq
2315743811:2315743826, ack 2921712201, win 51661, urg 256, length 15: HTTP
13:54:00.019650 IP 237.87.197.84.59664 > 127.0.0.1.80: Flags [FSRP.UEW], seq
1549881887:1549881898, ack 293323580, win 35237, urg 256, length 11: HTTP
13:54:00.019653 IP 234.63.134.74.44438 > 127.0.0.1.80: Flags [FSRP.UEW], seq
3492377168:3492377188, ack 2853225330, win 2995, urg 256, length 20: HTTP
13:54:00.019656 IP 211.183.245.26.47822 > 127.0.0.1.80: Flags [FSRP.UEW], seq
248707427:248707444, ack 1847235626, win 44472, urg 256, length 17: HTTP
13:54:25.913965 IP 35.69.179.41.44177 > 127.0.0.1.80: Flags [none], seq
1922026108:1922026113, win 52688, length 5: HTTP
13:54:25.913976 IP 60.174.254.110.40278 > 127.0.0.1.80: Flags [none], seq
941974136:941974137, win 15828, length 1: HTTP
13:54:25.913982 IP 228.133.240.28.37571 > 127.0.0.1.80: Flags [none], seq
2598328699:2598328707, win 2995, length 8: HTTP
```

Figure 9: TCP Xmas Flood with controlled TCP Flags

Layer 7 attack payloads are split into GET/HEAD and POST Floods. With each type, the malware iterates over a predefined list of 89 static User-Agent strings (full list provided in Figure 13). While the User-agents aren't under attacker control, the HTTP floods do allow

the attack to control the name of parameters, create additional GET parameter, change the length of parameters or add a randomized value that will be injected into said parameter. This last parameter's value will always be numeric.

```
14:05:57.443855 IP 127.0.0.1.38405 > 127.0.0.1.80: Flags [P.], seq 1:164, ack 1,
win 342, options [nop,nop,TS val 27536598 ecr 27536598], length 163: HTTP: GET
/?arg=59700620 HTTP/1.1
E.....@.@..y...........Px;.v..G=...V.......
..,...,.GET /?arg=59700620 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.97 Safari/537.36

14:05:51.766135 IP 127.0.0.1.38399 > 127.0.0.1.80: Flags [P.], seq 1:190, ack 1,
win 342, options [nop,nop,TS val 27535179 ecr 27535179], length 189: HTTP: GET
/?param=69403809 HTTP/1.1
E.....@.@.~............P.....J.....V.......
..'K..'KGET /?param=69403809 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (iPad; CPU OS 9_2_1 like Mac OS X) AppleWebKit/601.1.46 (KHTML,
like Gecko) Version/9.0 Mobile/13D15 Safari/601.1
```

Figure 10: HTTP GET Flood

```
14:06:36.072700 IP 127.0.0.1.38413 > 127.0.0.1.80: Flags [P.], seq 1:189, ack 1,
win 342, options [nop,nop,TS val 27546255 ecr 27546255], length 188: HTTP: HEAD
/?param=7360591664 HTTP/1.1
E...>.@.@..C...........PX.lPK.$0...V.......
..R...R.HEAD /?param=7360591664 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML,
like Gecko) Version/9.0.2 Safari/601.3.9

14:06:38.599561 IP 127.0.0.1.38417 > 127.0.0.1.80: Flags [P.], seq 1:189, ack 1,
win 342, options [nop,nop,TS val 27546887 ecr 27546887], length 188: HTTP: HEAD
/?param=4211130389 HTTP/1.1
E...n.@.@..y...........P..;...6....V.......
..U...U.HEAD /?param=4211130389 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/601.4.4 (KHTML,
like Gecko) Version/9.0.3 Safari/601.4.4
```

Figure 11: HTTP HEAD Flood

```
14:07:17.003783 IP 127.0.0.1.38499 > 127.0.0.1.80: Flags [P.], seq 1:264, ack 1,
win 342, options [nop,nop,TS val 27556488 ecr 27556488], length 263: HTTP: POST
/?param=8478584321 HTTP/1.1
E..;Bf@.@..T.........c.P).8.0N.a...V./.....
..z...z.POST /?param=8478584321 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0
Connection: keep-alive
Keep-Alive: 119
Content-Length: 8478
Content-Type: application/x-www-form-urlencoded
```

Figure 12: HTTP POST Flood

A noticeable characteristic of the HTTP POST Flood includes an attacker-controlled parameter that appears in GET parameters. These requests don't actually include any POST data as part of the request, even when utilizing the custom parameter and randomized value injection features.

```
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106
Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko)
Version/9.0.2 Safari/601.3.9
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111
Safari/537.36
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111
Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Windows NT 6.3; WOW64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/601.4.4 (KHTML, like Gecko)
Version/9.0.3 Safari/601.4.4
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.3.9 (KHTML, like Gecko)
Version/9.0.2 Safari/601.3.9
```

Figure 13: List of used HTTP User-Agent Headers

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111
Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.97
Safari/537.36
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106
Safari/537.36
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.97
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586
Mozilla/5.0 (iPad; CPU OS 9_2 like Mac OS X) AppleWebKit/601.1.46 (KHTML, like Gecko)
Version/9.0 Mobile/13C75 Safari/601.1
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/601.2.7 (KHTML, like Gecko)
Version/9.0.1 Safari/601.2.7
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (X11; Linux x86_64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.4.4 (KHTML, like Gecko)
Version/9.0.3 Safari/601.4.4
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.82
Safari/537.36
Mozilla/5.0 (Windows NT 5.1; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0
Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.82
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_0) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.82
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/601.3.9 (KHTML, like Gecko)
Version/9.0.2 Safari/537.86.3
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/600.8.9 (KHTML, like Gecko)
Version/8.0.8 Safari/600.8.9
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu
Chromium/47.0.2526.106 Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.0; Trident/5.0; Trident/5.0)
```

Figure 13: List of used HTTP User-Agent Headers (continued)

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11) AppleWebKit/601.1.56 (KHTML, like Gecko)
Version/9.0 Safari/601.1.56
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; Trident/5.0)
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.97 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.97
Safari/537.36
Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.73
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/534.59.10 (KHTML, like Gecko)
Version/5.1.9 Safari/534.59.10
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/42.0.2311.135 Safari/537.36 Edge/12.10240
Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0
Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.5.0
Mozilla/5.0 (iPad; CPU OS 9_2_1 like Mac OS X) AppleWebKit/601.1.46 (KHTML, like Gecko)
Version/9.0 Mobile/13D15 Safari/601.1
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.82 Safari/537.36
Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0) Gecko/20100101 Firefox/44.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.97 Safari/537.36
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromi-
um/47.0.2526.73 Chrome/47.0.2526.73 Safari/537.36
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.90
Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.80
Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_0) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.111 Safari/537.36
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.97
Safari/537.36
Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:43.0) Gecko/20100101 Firefox/43.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:44.0) Gecko/20100101 Firefox/44.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.97 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/47.0.2526.106 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.75.14 (KHTML, like Gecko)
Version/7.0.3 Safari/7046A194A
Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106
Safari/537.36
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0
Safari/537.36
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:42.0) Gecko/20100101 Firefox/42.0
```

Figure 13: List of used HTTP User-Agent Headers (continued)

5.0 / Observed Campaigns / On June 7, 2016, Akamai's Security Operations Center detected and mitigated a single attack to one of our clients reaching almost 250 Gbps at its peak. The attack consisted of a UDP Flood with signatures that match the Kaiten/STD malware-generated traffic. Bandwidth statistics are shown in Figure 14, with sample traffic shown in Figure 15.

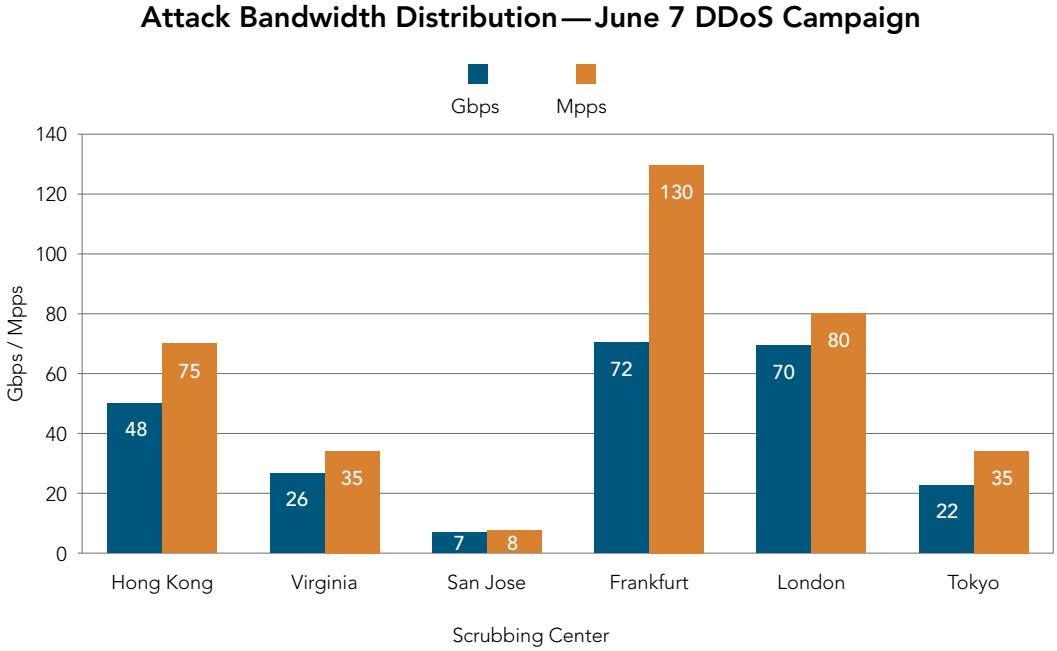

Figure 14: Attack bandwidth per Scrubbing Center

```
22:48:55.057876 IP xxx.xx.200.167.42170 > xxx.xxx.0.47.80: UDP, length 600
22:48:55.057880 IP xxx.xx.154.49.39923 > xxx.xxx.0.47.80: UDP, length 600
22:48:55.057902 IP xxx.xxx.229.79.59552 > xxx.xxx.0.47.80: UDP, length 600
22:48:55.057904 IP xxx.xxx.36.126.53227 > xxx.xxx.0.47.80: UDP, length 600
22:48:55.057905 IP xxx.xxx.31.173.56425 > xxx.xxx.0.47.80: UDP, length 600
22:48:55.057907 IP xxx.xxx.139.13.46283 > xxx.xxx.0.47.80: UDP, length 600
```

Figure 15: Sample malicious packets taken from DDoS campaign

On April 10, 2016, Akamai detected a layer-7 attack peaking at an estimated 500,000 requests per second. The distribution of this attack's bandwidth is provided in Figure 16.

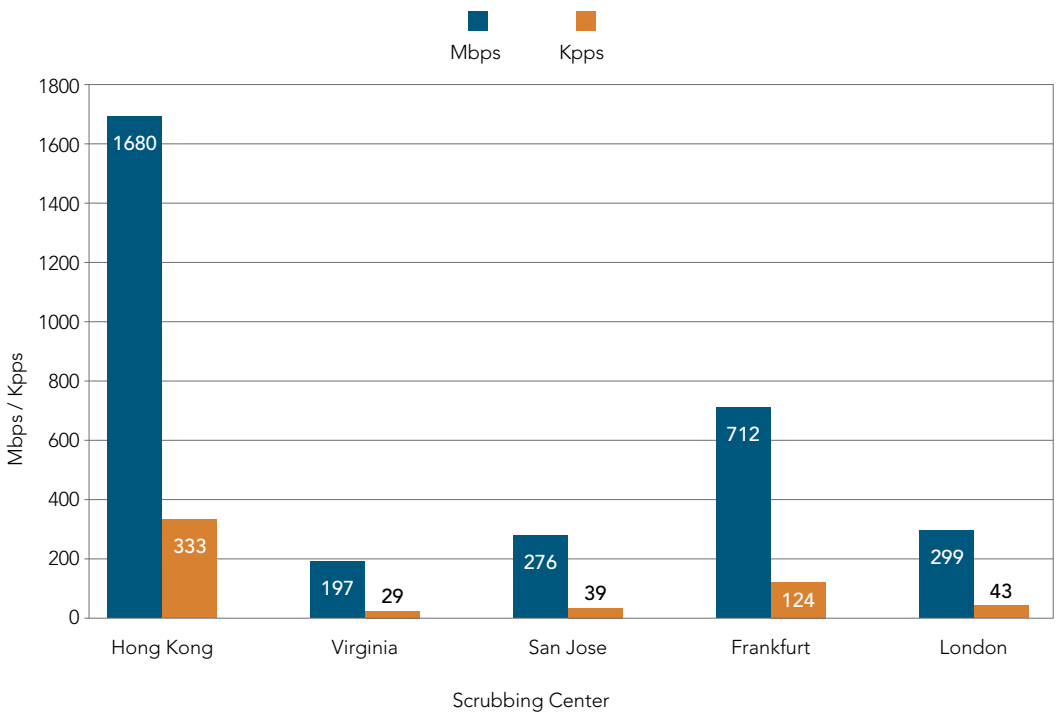

Figure 16: Attack bandwidth per Scrubbing Center

6.0 / Recommended Detection Methods / To detect an active infection an admin should check the crontab configuration for all user accounts on the suspect system. An example of an infected system's crontab is shown in Figure 17.

```
user@Ubuntu:/home/user/Desktop# crontab -l
* * * * * /home/user/Desktop/{malware filename} > /dev/null 2>&1 &
user@Ubuntu:/home/user/Desktop#
```

Figure 17: Crontab showing an active infection

Since the auto-start crontab entry gets inserted only on malware execution, the user can safely remove it and proceed with stopping the existing malicious running process.

The process is identified by the name **irq** and can be found using the **ps** utility as shown in Figure 18.

```
user@Ubuntu:/home/user/Desktop# ps -Fp $(pidof irq)
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
user     12103  2144 60    69   164   0 05:28 pts/7    04:29:51 irq
user@Ubuntu:/home/user/Desktop#
```

Figure 18: Identifying the malicious process

To identify that the malware sample is from the Kaiten/STD family, we have created the following Yara rule shown in Figure 19. It's configured to match on the two unique ASCII sentences found in the packed binary as well as the number of sections.

```
import "elf"

rule STD
{
   meta:
        author = "Akamai SIRT"
        description = "Kaiten/STD DDoS malware"

  strings:
        $s0 = "shitteru koto dake"
        $s1 = "nandemo wa shiranai wa yo,"

   condition:
        elf.number_of_sections == 0 and
        elf.number_of_segments == 2 and
        $s0 and $s1
}
```

Figure 19: Yara rule to detect Kaiten/STD malware

7.0 / Conclusion / Botnets built on the back of SOHO and IoT devices are steadily growing and can be leveraged in large-scale scanning, compromise systems through use of default names and passwords, and DDoS campaigns.

Some of these systems are easily compromised with publicly available exploits and knowledge. They can also be weaponized using publicly available attack toolkits and malware. These trends and tactics are unlikely to go away and the relative ease of building and renting these botnets will continue to lower the bar even further for attackers.

Owners should be periodically checking for and applying firmware updates. This will help keep long-lived and publicly accessible exploits from working against these devices by patching the flaws that introduced the exploitation opportunity.

It's also recommended that device owners disable unnecessary ports and services that potentially expose additional attack surface and ultimately present an exploitation opportunity. In some cases, these additional services can be leveraged by attackers for reflected DDoS campaigns without needing to actually exploit the machine. Their ability to leverage the device in these types of attacks is merely an underlying flaw with a particular service or protocol such as uPnP, mDNS, TFTP, etc.

Other measures, like changing default usernames and passwords, would appear to be common sense, yet devices with default credentials exist across the internet in many types (Routers, SCADA, IPcam, NAS, etc.) and are traditionally low-hanging fruit targeted by attackers. These shortcomings are usually exploited via large scale scanning campaigns using existing botnets and publicly available lists of known default, common, and weak credential pairs.

In closing, owners of the devices being targeted and exploited must stay vigilant and attempt to keep these systems up to date and hardened against these attacks. Network owners should research what they're deploying on their networks and take actionable steps to reduce known and unknown potential risks proactively. The basic defenses that protect networks from the known weaknesses of today are often the same as the defenses we need to protect them from the unknown attacks of tomorrow.