radware

IoT Attack Handbook

A Field Guide to Understanding IoT Attacks from the Mirai Botnet to Its Modern Variants

Ron Winward
Security Evangelist, Radware

SHARE THIS BROCHURE

IoT Attack Handbook

Contents

Mirai is an IoT botnet that was designed to exploit vulnerabilities in poorly secured IoT devices for use in large-scale DDoS attacks.

Released to the public by its author in 2016, it remains responsible for some of the most damaging and widely publicized DDoS attacks on the internet.

The threat of Mirai continues to change the security landscape in significant ways. When initially released, Mirai was a large botnet capable of incredibly large attacks because it maintained a large bot count. Today, Radware witnesses factions of smaller Mirai botnets rather than a few large ones. As more people understand how to run it, more people are competing for the pool of devices. In addition, botnets like BrickerBot and Hajime eliminate vulnerable devices from the available infection pool.

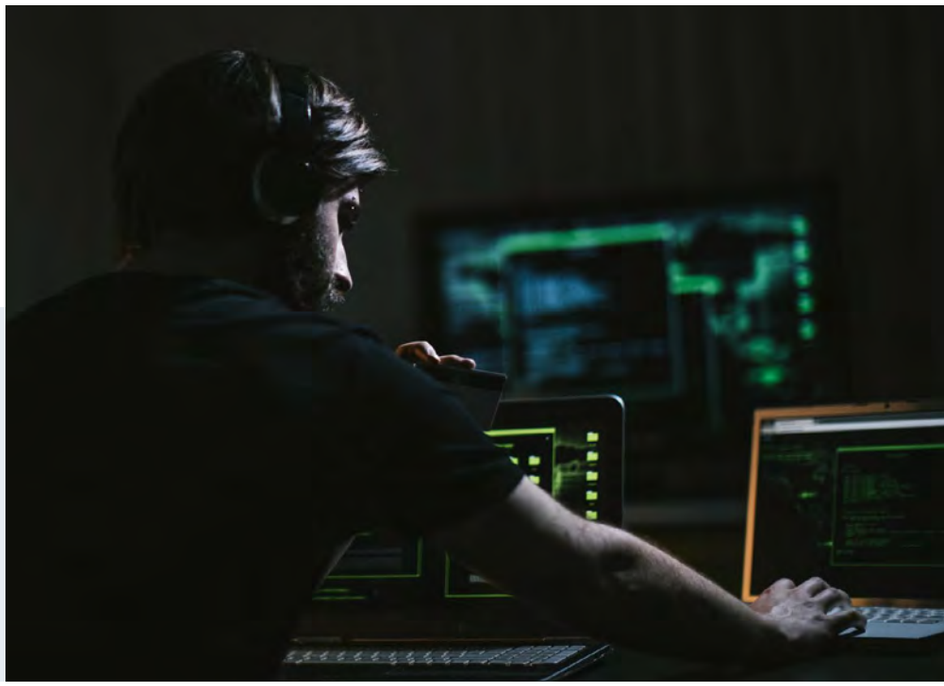
Perhaps the most compelling aspect of Mirai was the public release of the source code. Upon release, anyone, anywhere, could create their own botnet. With the source code available and instructions clearly documented, new threat actors had an existing framework that they could modify by adding code for new vectors or additional behavior to the botnets.

This has happened. Although Mirai is several years old now, it is still active in its original form in addition to modern variants. Botnets such as Masuta, Owari, DaddysMirai and Orion all include Mirai attack code. Evidence also suggests that other IoT botnets like IoT_Reaper/IoTroop and Satori are based on the Mirai framework, albeit different approaches.

Most importantly, Mirai underscores the potential of IoT as a DDoS attack tool and how vulnerable poorly secured devices are.

Not all IoT devices are susceptible to Mirai infection. In fact, it can be difficult to achieve and maintain infection of a device. Even if a device can be infected and join a Mirai botnet, many devices behave differently once enslaved. For example, some devices crash and reboot once they are issued an attack command, which flushes Mirai from its system. In another example, it was observed that certain variables, like source or destination IP, could not be accurately implemented by a specific device, thus sending the attack to the wrong destination.

This handbook is a study of the original Mirai attack vectors and some of its variants. Each of the attacks is dissected in a lab environment, allowing the reader to discover the flexibility of each one. The intent is for the reader to become familiar with each attack vector, understand its profile and parameters, and think about how to defend against each of these attacks if needed.



UNDER ATTACK

THE INFAMOUS MIRAI BOTNET *was responsible for some of the largest and most distributive DDoS attacks in recent history, including an attack against security journalist Brian Krebs' website, French web host OVH, and DNS provider Dyn. Commandeering hundreds of thousands of IoT devices, it sent shock waves through the security marketplace with traffic volumes exceeding 1Tbps.*

01

Mirai Overview

The original Mirai code includes 10 types of attacks, and each is configurable with several variables.

Many of these attack vectors are based on traditional DDoS attack types but have been customized and/or enhanced for use in Mirai.

There are two more attacks partially written in the code (Proxy and CFNull), but they were not finished.

The platform was written to allow for multitenancy and transactional access. Once the C2 server and botnet are established, additional users can be added to the platform. This means that public botnet access is as simple as a business transaction.

MIRAI ATTACK CLASSIFICATION

There are four classifications of attacks in the original source code.

1: TCP: *SYN, ACK, STOMP*

2: UDP: *UDP, VSE, DNS, UDPPLAIN*

3: GRE: *GREIP, GREETH*

4: APP: *HTTP*

```
greeth: GRE Ethernet flood
http: HTTP flood
dns: DNS resolver flood using the targets domain;
input IP is ignored
syn: SYN flood
greip: GRE IP flood
stomp: TCP stomp flood
udpplain: UDP flood with less options, optimized for higher PPS
udp: UDP flood
vse: Valve source engine specific flood
ack: ACK flood
```

Users accounts are added (figure 1) from the C2 console, and users are provided with several options, such as how many bots they may control, their maximum attack duration (seconds) and how long they must wait between attacks (seconds).

```
mirai-user@botnet# adduser
Enter new username: 3bots
Enter new password: 3bots
Enter wanted bot count (-1 for full net): 3
Max attack duration (-1 for none): 30
Cooldown time (0 for none): 30
New account info:
Username: 3bots
Password: 3bots
Bots: 3
Continue? (y/N)y
User added successfully.
mirai-user@botnet#
```

FIGURE 1: ADDING USERS ON THE MIRAI CONSOLE

LAB ATTACKERS

Raspberry Pi 3 devices running Raspbian (Stretch) were chosen as the Mirai botnet members for this analysis. Raspbian is a Debian-based version of Linux built for the Raspberry Pi. Neither Raspberry Pi nor Raspbian are known to be vulnerable to Mirai. Rather, Mirai was manually loaded onto the Pis in the lab and run in debug mode for the purpose of this research.

It was observed that IoT devices in the lab exhibited similar behavior to the Pis in terms of relative attack rates (BPS/PPS ratios for different vectors were similar), but the Raspberry Pi 3s were more flexible in a lab environment and created much larger attacks per device. Not only are the Pis more powerful than other lab IoT devices because of RAM and CPU, but they also don't crash unexpectedly like IoT devices can.

Execution is the same with the Raspberry Pi devices in this environment as with IoT devices in the wild, with the exception that the loader process was not used to infect the Pis. Instead, Mirai is manually executed upon startup. Nevertheless, the Pi devices are under the control of the C2 server and attacks are launched from the C2 server just as they are in the wild.

This analysis also includes a brief comparison of the Mirai and Owari botnet attacks. The Owari tests were run from real IP cameras under the control of a lab Owari botnet. Owari code does not include a debug mode, and it was not considered an accurate comparison to run individually compiled Owari attacks on the Pis. When comparing C2-based attacks to manually compiled attacks, the manual attacks are more aggressive and thus not a fair comparison.

THREAT RANKING

In the following analysis, each Mirai vector is sequentially ranked against its counterparts. The method for the score was based on attack velocity (BPS, PPS or both), its default behavior (an attack that is not specifically crafted), and other factors.

The attacks are individually ranked on a scale from 10 to one (10 is the most threatening attack). The ranking does not necessarily imply that a score of 10 is a significantly greater threat than a score of one, but rather that each attack is a significant threat that has been crafted to achieve a significant, custom result.

The score for each Mirai vector is based on the Pi bot behavior in the lab rather than IoT devices. Sometimes IoT devices exhibited different behavior than the Pi bots, but overall behavior is similar.

It is known that servers are also used as Mirai bots in the wild, similar to how Pis were used in the lab for this analysis. Servers are clearly more capable and dangerous, so the Pi behavior compared to IoT device behavior is considered relevant.

The attacks are individually ranked on a scale from 10 to one (10 is the most threatening attack).

02

Mirai Attack Vectors

DNS

CHARACTERISTICS

PROTOCOL: UDP

BANDWIDTH PROFILE: Medium BPS, High PPS

PACKET SIZE: Small (93 bytes)

NOTES: Very difficult attack to defend against without specific tools

THREAT RANKING: **10**

The DNS attack included in Mirai is an interesting attack and probably the most notable due to the high-profile attacks it was used in. Mirai is not the first time this attack was seen in the wild, but it is likely that the first time it was included was in an IoT botnet, which added critical damage potential. Figure 2 shows the control parameters of the attack.

Notice that the attacker must specify the domain being attacked. In this attack, the IP address specified in the syntax isn't even used. Instead, the bot generates a DNS query flood at the domain specified.

The attack itself is a query flood of random subdomains within the specified domain, in the format of $STRING.domain.com.

The IoT device sends this request to its local recursive DNS server. The request is likely hitting something inside of its local network first. Perhaps it's a local recursive server or maybe it's a local router that's also proxying DNS requests.

FIGURE 2: MIRAI CONSOLE FOR THE DNS ATTACK

```
mirai-user@botnet# dns 192.168.3.10 120 domain=example.com ?
List of flags key=val seperated by spaces. Valid flags for this method are

tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random
domain: Domain name to attack
dhid: Domain name transaction ID, default is random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

The device floods the DNS server with A-record lookups for $STRING.domain.com. In the example below, devices are flooding 192.168.3.4 with lookups for $STRING.example.com.

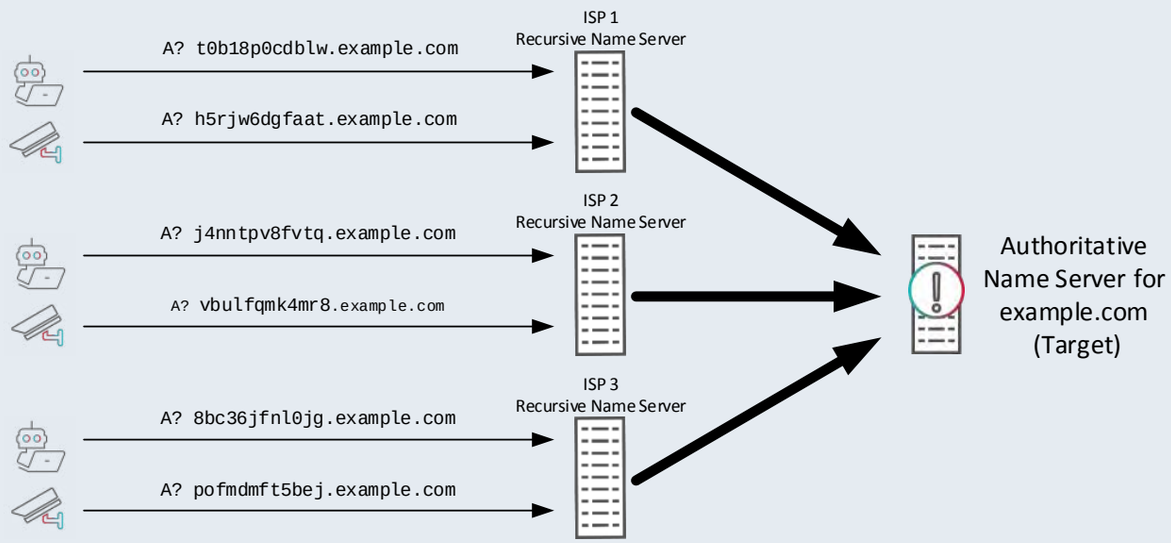
```
11:10:43.697367 IP 192.168.3.114.34569 > 192.168.3.4.53: 11725+ A? t0b18p0cdblw.example.com. (45)
11:10:43.698271 IP 192.168.3.112.41054 > 192.168.3.4.53: 18327+ A? h5rjw6dgfaat.example.com. (45)
11:10:43.698885 IP 192.168.3.115.57475 > 192.168.3.4.53: 951+ A? j4nntpv8fvtq.example.com. (45)
11:10:43.700153 IP 192.168.3.115.62723 > 192.168.3.4.53: 52021+ A? vbulfqmk4mr8.example.com. (45)
11:10:43.700775 IP 192.168.3.114.37722 > 192.168.3.4.53: 21143+ A? 8bc36jfnl0jg.example.com. (45)
11:10:43.701182 IP 192.168.3.113.14438 > 192.168.3.4.53: 65156+ A? pofmdmft5bej.example.com. (45)
```

The local server doesn't have this record cached, so it forwards the lookup request to the authoritative name server for the domain. In this example, 192.168.2.53 is the authoritative name server for domain "example.com," so 192.168.3.4 forwards the request to it.

```
11:09:47.241041 IP 192.168.3.4.57781 > 192.168.2.53.53: 63948+% [1au] A? qp17vht88mgj.example.com. (56)
11:09:47.241168 IP 192.168.3.4.49960 > 192.168.2.53.53: 5832+% [1au] A? 56j7gru4r368.example.com. (56)
11:09:47.241553 IP 192.168.3.4.51086 > 192.168.2.53.53: 54527+% [1au] A? lkark163wqlo.example.com. (56)
11:09:47.241762 IP 192.168.3.4.34301 > 192.168.2.53.53: 37060+% [1au] A? dlg5uwouw7lb.example.com. (56)
11:09:47.241998 IP 192.168.3.4.51425 > 192.168.2.53.53: 13561+% [1au] A? 8prf8ffbt7qh.example.com. (56)
11:09:47.242219 IP 192.168.3.4.38028 > 192.168.2.53.53: 57721+% [1au] A? oeinrrct1bc5.example.com. (56)
```

This is where the attack scales incredibly. Suddenly, the authoritative DNS server for the domain is under a flood of queries from real DNS servers on the internet. Can the victim simply block all DNS requests from real servers around the internet? Not if they want to stay online for legitimate queries.

FIGURE 3: TOPOLOGY OF THE MIRAI DNS ATTACK



CAUTION

The DNS attack is a very high packets per second (PPS) attack. Under load, you can expect it to exhaust link throughput from PPS before it exhausts bits per second (BPS). Be sure to monitor the PPS rate if you see this attack because if you are only monitoring BPS, it might appear that you have more available bandwidth than you actually do.

This vector is very difficult to defend against without the appropriate tools. An organization cannot simply block port 53 and cannot block the source IP. It requires getting inside the payload of the query and mitigating the attack from there by blocking only the randomized subdomains.

VSE

CHARACTERISTICS

PROTOCOL: UDP

BANDWIDTH PROFILE: Medium BPS, High PPS

PACKET SIZE: Small (67 bytes)

NOTES: Built to target game platforms; however, it is a high PPS attack that could be used on other targets

THREAT RANKING: 9

There is a fascinating relationship with online gaming and DDoS. Online gaming has long been at the center of attacks, and it can be argued that the proliferation of DDoS tools and techniques can be somewhat attributed to this space. Simply put, people have been regularly and creatively attacking game servers for years.

The Valve Source Engine attack is specially crafted for servers that run certain games from the developer Valve Corporation. These online multiplayer games are built on a software framework (or a game engine) called "Source," which is responsible for things like rendering images, sounds, gameplay, networking, etc.

A Valve game server running the Source engine is specifically vulnerable to this attack. Valve Corporation is responsible for a number of well-known games, including *Half-Life, Team Fortress 2*, and *Counter-Strike: Global Offensive*, as well as others that run the Source engine.[1]

Anyone can run servers for these games, so it's not an attack on Valve themselves. In the example below, you can see a game client connect to a Valve game server on a private LAN. When the game server starts, it listens for player connections on UDP port 27015.

```
$ netstat -ln | grep 27015
tcp        0      0 127.0.1.1:27015      0.0.0.0:*               LISTEN
udp        0      0 0.0.0.0:27015        0.0.0.0:*
$
```

New players wanting to connect to the server will do so on UDP/27015.

```
21:12:47.027872 IP 192.168.1.252.60814 > 192.168.3.11.27015: UDP, length 25
21:12:47.028006 IP 192.168.1.252.60815 > 192.168.3.11.27015: UDP, length 25
21:12:47.028019 IP 192.168.1.252.60816 > 192.168.3.11.27015: UDP, length 25
21:12:47.028028 IP 192.168.1.252.60817 > 192.168.3.11.27015: UDP, length 25
```

[1] *https://en.wikipedia.org/wiki/Source_(game_engine)#Games_using_Source*

All subsequent traffic happens on this predetermined port throughout the game. The problem is that these servers must listen on their defined port and will usually allow incoming connections from any IP address. It's true that the server operator can change the port, but those attacking the network likely already know which port the game is running on, so they can simply change the attack destination.

The result is that it's difficult to distinguish between legitimate UDP connection attempts and illegitimate ones. While the server can whitelist or only permit specific IP addresses if desired, IoT-based DDoS attacks will quickly overrun a server with volume and packets per second (PPS), rendering it useless if unprotected upstream. Here are the control parameters of the attack:

FIGURE 4:
MIRAI CONSOLE FOR THE VSE ATTACK

```
mirai-user@botnet# vse 192.168.3.11 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

With the most basic syntax, the attack looks like this:

```
14:36:33.888087 IP 192.168.3.115.64420 > 192.168.3.10.27015: UDP, length 25
14:36:33.888469 IP 192.168.3.113.30051 > 192.168.3.10.27015: UDP, length 25
14:36:33.888839 IP 192.168.3.112.49586 > 192.168.3.10.27015: UDP, length 25
14:36:33.889197 IP 192.168.3.115.4447 > 192.168.3.10.27015: UDP, length 25
14:36:33.889551 IP 192.168.3.114.29292 > 192.168.3.10.27015: UDP, length 25
```

This is identical to real queries when clients join the game because it's a real Source Engine Query that is generated by the attacker.

FIGURE 5:
VSE ATTACK PAYLOAD

```
0000  00 0c 29 db 18 08 b8 27  eb d6 07 3f 08 00 45 00   ··)····' ···?··E·
0010  00 35 f1 b1 00 00 40 11  01 38 c0 a8 03 73 c0 a8   ·5····@· ·8···s··
0020  03 0b fc 72 69 87 00 21  53 29 ff ff ff ff 54 53   ···ri··! S)····TS
0030  6f 75 72 63 65 20 45 6e  67 69 6e 65 20 51 75 65   ource En gine Que
0040  72 79 00                                          ry·
```

CAUTION

The VSE attack is a very high packets per second (PPS) attack. Under load, it can be expected to exhaust link throughput from PPS before it exhausts bits per second (BPS). Monitor the PPS rate if you see this attack because if you are only monitoring BPS, it might appear that you have more available bandwidth than you actually do.

STOMP

CHARACTERISTICS

PROTOCOL: TCP

BANDWIDTH PROFILE: High BPS (highest of all Pi bot attacks), Low PPS

PACKET SIZE: Largest of all vectors (822 bytes)

NOTES: Default behavior creates an out-of-state condition

THREAT RANKING: **8**

The TCP STOMP attack is one of the more interesting attacks in Mirai. In fact, it was reportedly designed to defeat certain DDoS mitigation techniques because it is an in-session attack.

The intent of the attack is to establish a three-way TCP handshake, after which the attacking nodes send an ACK Flood within their session that has already been whitelisted by protection sets.

Here are the control parameters of the attack:

FIGURE 6: MIRAI CONSOLE FOR THE STOMP ATTACK

```
mirai-user@botnet# stomp 192.168.3.10 120 dport=80 ?
List of flags key=val seperated by spaces. Valid flags for this method are

len: Size of packet data, default is 512 bytes
rand: Randomize packet data content, default is 1 (yes)
tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
dport: Destination port, default is random
urg: Set the URG bit in IP header, default is 0 (no)
ack: Set the ACK bit in IP header, default is 0 (no) except for ACK flood
psh: Set the PSH bit in IP header, default is 0 (no)
rst: Set the RST bit in IP header, default is 0 (no)
syn: Set the ACK bit in IP header, default is 0 (no) except for SYN flood
fin: Set the FIN bit in IP header, default is 0 (no)

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

The session setup looks like this, with a three-way handshake between 192.168.3.111 (attacker) and 192.168.3.10 (target).

```
20:21:50.255975 IP 192.168.3.111.50114 > 192.168.3.10.80: Flags [S], seq 1984590386, win 29200, options [mss 1460,sackOK,TS val 17639602 ecr 0,nop,wscale 7], length 0
20:21:50.256113 IP 192.168.3.10.80 > 192.168.3.111.50114: Flags [S.], seq 518524438, ack 1984590387, win 28960, options [mss 1460,sackOK,TS val 17639394 ecr
  17639602,nop,wscale 7], length 0
20:21:50.256763 IP 192.168.3.111.50114 > 192.168.3.10.80: Flags [.], ack 1, win 229, options [nop,nop,TS val 17639602 ecr 17639394], length 0
```

The very next packet in the transaction is the attacker trying to send a PSH+ACK flood to the target.

```
20:21:50.257249 IP 192.168.3.111.40933 > 192.168.3.10.80: Flags [P.], seq 2481258496:2481259252, ack 467992576, win 42962, options [[bad opt]
20:21:50.257304 IP 192.168.3.10.80 > 192.168.3.111.40933: Flags [R], seq 467992576, win 0, length 0
20:21:50.257497 IP 192.168.3.111.40933 > 192.168.3.10.80: Flags [P.], seq 65536:66292, ack 1, win 42962, options [[bad opt]
20:21:50.257536 IP 192.168.3.10.80 > 192.168.3.111.40933: Flags [R], seq 467992576, win 0, length 0
20:21:50.257631 IP 192.168.3.111.40933 > 192.168.3.10.80: Flags [P.], seq 131072:131828, ack 1, win 42962, options [[bad opt]
20:21:50.257664 IP 192.168.3.10.80 > 192.168.3.111.40933: Flags [R], seq 467992576, win 0, length 0
20:21:50.257884 IP 192.168.3.111.40933 > 192.168.3.10.80: Flags [P.], seq 196608:197364, ack 1, win 42962, options [[bad opt]
20:21:50.257920 IP 192.168.3.10.80 > 192.168.3.111.40933: Flags [R], seq 467992576, win 0, length 0
```

However, notice that the attacker switched to the source port of 40933 once the three-way handshake was complete. Although it is intended to be an in-session attack, this is technically out-of-state, and an OOS protection should detect this anomaly. Remember that the Mirai code is easily modifiable, and this behavior can change.

It should be noted that the original Mirai code does not allow for a source port to be defined, so unless the code is modified, OOS protections should defeat this attack.

```
mirai-user@botnet# stomp 192.168.3.10 120 sport=1234
Invalid flag key sport, near sport=1234
mirai-user@botnet#
```

Finally, this is an incredibly high BPS attack (the highest BPS using Raspberry Pi bots). This attack can easily threaten a target with volumetric capacity alone.

GREETH

CHARACTERISTICS

PROTOCOL: GRE

BANDWIDTH PROFILE: Profile: High BPS, Medium PPS

PACKET SIZE: Medium (592 bytes)

NOTES: Payload is Layer 2 Ethernet frames (Transparent Ethernet Bridging)

THREAT RANKING: 7

The GREETH attack is interesting and also mysterious. The payload of the attack includes Transparent Ethernet Bridging over GRE-encapsulated packets. The behavior of the attack is similar to the GREIP attack, but it also includes an L2 frame.

Here are the control parameters of the attack:

FIGURE 7: MIRAI CONSOLE FOR THE GREETH ATTACK

```
mirai-user@botnet# greeth 192.168.3.10 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

len: Size of packet data, default is 512 bytes
rand: Randomize packet data content, default is 1 (yes)
tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random
gcip: Set internal IP to destination ip, default is 0 (no)
source: Source IP address, 255.255.255.255 for random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
mirai-user@botnet# greeth 192.168.3.10 120
```

The control parameters are the same as GREIP, and despite the attack including a L2 payload, the attacker doesn't have control over the L2 contents.

On its own, it doesn't look much different in a TCPdump than the GREIP attack, aside from the slightly larger packet length (558 bytes instead of 544 bytes in GREIP).

```
22:00:37.053928 IP 192.168.3.115 > 192.168.3.10: GREv0, length 558: IP 9.216.137.67.30513 >
  248.6.163.90.16375: UDP, length 512
22:00:37.054838 IP 192.168.3.112 > 192.168.3.10: GREv0, length 558: IP 201.149.87.116.24440 >
  54.45.106.42.19356: UDP, length 512
22:00:37.055294 IP 192.168.3.113 > 192.168.3.10: GREv0, length 558: IP 54.228.144.26.59862 >
  97.157.49.23.27788: UDP, length 512
22:00:37.055747 IP 192.168.3.114 > 192.168.3.10: GREv0, length 558: IP 47.207.20.146.40931 >
  42.71.229.14.52950: UDP, length 512
22:00:37.068051 IP 192.168.3.111 > 192.168.3.10: GREv0, length 558: IP 137.112.84.84.29 >
  162.123.255.65.13701: UDP, length 512
```

Inspection of the complete payload is required to understand what's included.
This is the outer Layer 2 info (the lab bots attacking their target).

```
Ethernet II, Src: Raspberr _ d6:27:bf (b8:27:eb:d6:27:bf), Dst: AsustekC _ 42:fb:d2
  (2c:4d:54:42:fb:d2)
  Destination: AsustekC _ 42:fb:d2 (2c:4d:54:42:fb:d2)
      Address: AsustekC _ 42:fb:d2 (2c:4d:54:42:fb:d2)
   .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
   Source: Raspberr _ d6:27:bf (b8:27:eb:d6:27:bf)
      Address: Raspberr _ d6:27:bf (b8:27:eb:d6:27:bf)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
   Type: IPv4 (0x0800)
```

This is the outer Layer 3 info (the lab bots attacking their target).

```
Internet Protocol Version 4, Src: 192.168.3.111, Dst: 192.168.3.10
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   0000 00.. = Differentiated Services Codepoint: Default (0)
   .... ..00 = Explicit Congestion Notification: Not ECN-Capable
  Transport (0)
  Total Length: 578
  Identification: 0x62e2 (25314)
  Flags: 0x02 (Don't Fragment)
   0... .... = Reserved bit: Not set
   .1.. .... = Don't fragment: Set
   ..0. .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 64
  Protocol: Generic Routing Encapsulation (47)
  Header checksum: 0x4de1 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.3.111
  Destination: 192.168.3.10
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
```

This is the beginning of the encapsulated GRE payload.

```
Generic Routing Encapsulation (Transparent Ethernet bridging)
  Flags and Version: 0x0000
   0... .... .... .... = Checksum Bit: No
       .0.. .... .... .... = Routing Bit: No
       ..0. .... .... .... = Key Bit: No
       ...0 .... .... .... = Sequence Number Bit: No
       .... 0... .... .... = Strict Source Route Bit: No
   .... .000 .... .... = Recursion control: 0
   .... .... 0000 0... = Flags (Reserved): 0
   .... .... .... .000 = Version: GRE (0)
  Protocol Type: Transparent Ethernet bridging (0x6558)
```

This is the inner Layer 2 frame.

```
Ethernet II, Src: a2:81:e5:25:e7:d2 (a2:81:e5:25:e7:d2), Dst: a0:25:58:f9:6f:f5 (a0:25:58:f9:6f:f5)
  Destination: a0:25:58:f9:6f:f5 (a0:25:58:f9:6f:f5)
        Address: a0:25:58:f9:6f:f5 (a0:25:58:f9:6f:f5)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: a2:81:e5:25:e7:d2 (a2:81:e5:25:e7:d2)
   Address: a2:81:e5:25:e7:d2 (a2:81:e5:25:e7:d2)
   .... ..1. .... .... .... .... = LG bit: Locally administered address
       (this is NOT the factory default)
   .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
```

This is the inner Layer 3 packet.

```
Internet Protocol Version 4, Src: 137.112.84.84, Dst: 44.94.226.208
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   0000 00.. = Differentiated Services Codepoint: Default (0)
   .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 540
  Identification: 0x8521 (34081)
  Flags: 0x02 (Don't Fragment)
   0... .... = Reserved bit: Not set
   .1.. .... = Don't fragment: Set
   ..0. .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 64
  Protocol: UDP (17)
  Header checksum: 0xc6bc [validation disabled]
  [Header checksum status: Unverified]
  Source: 137.112.84.84
  Destination: 44.94.226.208
   [Source GeoIP: Unknown]
   [Destination GeoIP: Unknown]
```

This is the inner Layer 4 protocol.

```
User Datagram Protocol, Src Port: 62274, Dst Port: 62501
  Source Port: 62274
  Destination Port: 62501
  Length: 520
  Checksum: 0x3acf [unverified]
  [Checksum Status: Unverified]
  [Stream index: 20]
```

The source and destination MAC addresses are randomly generated and are sometimes malformed or illegal.

The GREETH attack is an extremely high velocity attack, both in PPS and BPS. In the lab, it's slightly less PPS than the GREIP attack because of the larger packets being generated, but it is ranked higher because of the overall BPS and larger packet size.

CAUTION

Remember that GRE is its own protocol. If you're using firewall filters or access lists to help identify or block the traffic, remember that it's not TCP or UDP, and you might need to account for GRE specifically in your policy.

GREIP

CHARACTERISTICS

PROTOCOL: GRE

BANDWIDTH PROFILE: Profile: High BPS, Medium PPS

PACKET SIZE: Medium (578 bytes)

NOTES: In the wild, this can be a very high PPS attack

THREAT RANKING: **6**

The GREIP attack is an incredibly interesting and versatile attack. GRE is an attack payload that became popular with Mirai. Before Mirai it was not a common vector. It is particularly interesting because of its flexibility and velocity, and it can be confusing if you haven't seen it before.

Here are the control parameters of the attack:

FIGURE 8: MIRAI CONSOLE FOR THE GREIP ATTACK

```
mirai-user@botnet# greip 192.168.3.10 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

len: Size of packet data, default is 512 bytes
rand: Randomize packet data content, default is 1 (yes)
tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random
gcip: Set internal IP to destination ip, default is 0 (no)
source: Source IP address, 255.255.255.255 for random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

The attack encapsulates a 512-byte payload inside of a GRE packet. By default, the inner packet has a random source and destination IP, as well as source and destination ports.

```
21:12:56.810870 IP 192.168.3.114 > 192.168.3.10: GREv0, length 544: IP 124.209.16.45.13109 > 71.2.185.186.8696: UDP, length 512
21:12:56.811218 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.55994 > 64.218.22.38.45651: UDP, length 512
21:12:56.811567 IP 192.168.3.112 > 192.168.3.10: GREv0, length 544: IP 9.221.112.93.52273 > 161.174.89.139.18817: UDP, length 512
21:12:56.811951 IP 192.168.3.113 > 192.168.3.10: GREv0, length 544: IP 68.60.217.207.37360 > 250.178.96.17.11508: UDP, length 512
21:12:56.812797 IP 192.168.3.114 > 192.168.3.10: GREv0, length 544: IP 124.209.16.45.13003 > 109.94.137.200.53337: UDP, length
21:12:56.813713 IP 192.168.3.112 > 192.168.3.10: GREv0, length 544: IP 9.221.112.93.24496 > 76.127.227.47.19649: UDP, length 512
```

Upon closer inspection, the actual source IP address is consistent with the source host. It's a random IP, but it's consistently repeated by the host. For example, inspect attacker 192.168.3.111. It is sending an encapsulated source IP of 137.112.84.84.

```
02:16:42.621079 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.47934 > 128.52.8.101.5061: UDP, length 512
02:16:42.621532 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.2930 > 181.88.46.255.61256: UDP, length 512
02:16:42.621988 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.9304 > 34.148.195.229.46077: UDP, length 512
02:16:42.622445 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.10849 > 38.191.235.82.63822: UDP, length 512
02:16:42.622899 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.1102 > 98.196.17.168.2268: UDP, length 512
02:16:42.623356 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.53240 > 74.23.39.146.52563: UDP, length 512
02:16:42.623813 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.46002 > 26.80.150.99.27450: UDP, length 512
```

Note the control parameters in the CLI screenshot in Figure 8, where you can manipulate the attack. In the example below, we've sent an inside destination IP to match the outside (192.168.3.10) and we've set the destination port to 1234.

```
19:05:56.577323 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 183.183.1.173.52012 > 192.168.3.10.1234: UDP, length 512
19:05:56.577771 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 183.183.1.173.16397 > 192.168.3.10.1234: UDP, length 512
19:05:56.578219 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 183.183.1.173.47626 > 192.168.3.10.1234: UDP, length 512
19:05:56.578667 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 183.183.1.173.21501 > 192.168.3.10.1234: UDP, length 512
19:05:56.579116 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 183.183.1.173.61218 > 192.168.3.10.1234: UDP, length 512
19:05:56.579565 IP 192.168.3.111 > 192.168.3.10: GREv0, length 544: IP 183.183.1.173.52420 > 192.168.3.10.1234: UDP, length 512
```

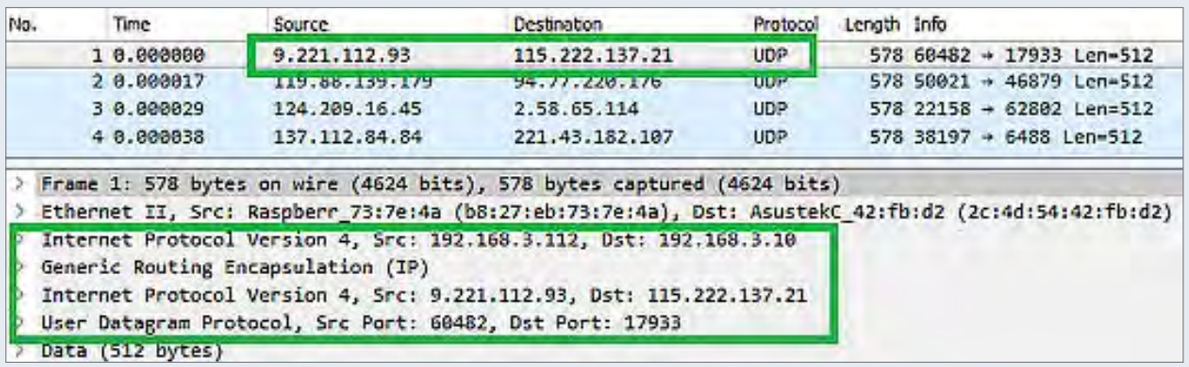
The control parameters allow specification or randomization of the source IP address, but we find that this doesn't work correctly. Only the first octet is generated and the last three are zeros.

```
mirai-user@botnet# greip 192.168.3.10 30 source=255.255.255.255

02:22:02.438705 IP 255.0.0.0 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.11402 > 224.87.67.239.62801: UDP, length 512
02:22:02.439158 IP 255.0.0.0 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.11619 > 208.124.252.238.830: UDP, length 512
02:22:02.439614 IP 255.0.0.0 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.49475 > 46.7.80.97.57894: UDP, length 512

mirai-user@botnet# greip 192.168.3.10 30 source=10.20.30.40

02:49:44.780489 IP 10.0.0.0 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.29302 > 204.13.165.8.49989: UDP, length 512
02:49:44.780946 IP 10.0.0.0 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.12716 > 199.86.119.209.4526: UDP, length 512
02:49:44.781402 IP 10.0.0.0 > 192.168.3.10: GREv0, length 544: IP 137.112.84.84.42846 > 46.127.141.138.14333: UDP, length 512
```

It's also worth mentioning that the tool used to decode this payload may present it differently. In the examples so far, we have used TCPdump for captures. But if you bring a capture into a different protocol analyzer like Wireshark, you might see only the inner packets in the summary. Here is an example from Wireshark, which shows the inside packets instead of the outside.

Finally, and most important, it's an incredibly powerful attack and one of the primary reasons why GRE attacks have been in the news since Mirai was released.

FIGURE 9: WIRESHARK RENDERING OF THE GREIP ATTACK PAYLOAD SHOWING INNER PACKET



CAUTION

Remember that GRE is its own protocol. If you're using firewall filters or access lists to help identify or block the traffic, remember that it's not TCP or UDP, and you might need to account for GRE specifically in your policy.

SYN

CHARACTERISTICS

PROTOCOL: TCP

BANDWIDTH PROFILE: Moderate BPS, High PPS

PACKET SIZE: Small (74 bytes)

NOTES: Volumetric attributes of attack can overwhelm devices, even if they have SYN Flood protection

THREAT RANKING: 5

The Mirai SYN attack is a classic SYN Flood designed to exploit the TCP stack of its target. The default parameters are to have random source port and random destination port, but to be effective, an attacker will likely set a target destination port when targeting an application.

Here are the control parameters of the attack:

FIGURE 10: MIRAI CONSOLE FOR THE SYN ATTACK

```
mirai-user@botnet# syn 192.168.3.10 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random
urg: Set the URG bit in IP header, default is 0 (no)
ack: Set the ACK bit in IP header, default is 0 (no) except for ACK flood
psh: Set the PSH bit in IP header, default is 0 (no)
rst: Set the RST bit in IP header, default is 0 (no)
syn: Set the ACK bit in IP header, default is 0 (no) except for SYN flood
fin: Set the FIN bit in IP header, default is 0 (no)
seqnum: Sequence number value in TCP header, default is random
acknum: Ack number value in TCP header, default is random
source: Source IP address, 255.255.255.255 for random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

Notice the inclusion of the six original TCP control flags (URG, ACK, PSH, RST, SYN and FIN) as attack parameters. Attackers have more control and options with the TCP attacks in Mirai. With the most basic syntax, the attack looks like this:

```
16:05:22.345298 IP 192.168.3.113.12814 > 192.168.3.10.1205: Flags [S], seq 2443004405, win 0, options
  [mss 1415,sackOK,TS val 2843724142 ecr 0,nop,wscale 6], length 0
16:05:22.345504 IP 192.168.3.115.60205 > 192.168.3.10.43834: Flags [S], seq 3654811864, win 0, options
  [mss 1404,sackOK,TS val 742027894 ecr 0,nop,wscale 6], length 0
16:05:22.345708 IP 192.168.3.112.44190 > 192.168.3.10.44009: Flags [S], seq 1531163422, win 0, options
  [mss 1404,sackOK,TS val 194787267 ecr 0,nop,wscale 6], length 0
16:05:22.345916 IP 192.168.3.114.35793 > 192.168.3.10.51112: Flags [S], seq 135361215, win 0, options
  [mss 1408,sackOK,TS val 1457994201 ecr 0,nop,wscale 6], length 0
16:05:22.346123 IP 192.168.3.111.58524 > 192.168.3.10.13939: Flags [S], seq 2866898217, win 0, options
  [mss 1402,sackOK,TS val 1982980123 ecr 0,nop,wscale 6], length 0
```

Setting the source IP to random does work with this vector.

```
15:49:22.197799 IP 229.82.235.233.16607 > 192.168.3.10.1469: Flags [S], seq 1991290348, win 0, options
  [mss 1402,sackOK,TS val 1982980123 ecr 0,nop,wscale 6], length 0
15:49:22.197997 IP 156.237.150.25.27113 > 192.168.3.10.36815: Flags [S], seq 3139468395, win 0, options
  [mss 1402,sackOK,TS val 1982980123 ecr 0,nop,wscale 6], length 0
15:49:22.198275 IP 154.172.179.90.7213 > 192.168.3.10.1464: Flags [S], seq 3876991215, win 0, options
  [mss 1402,sackOK,TS val 1982980123 ecr 0,nop,wscale 6], length 0
15:49:22.198485 IP 239.181.99.232.56244 > 192.168.3.10.55042: Flags [S], seq 3440508772, win 0, options
  [mss 1402,sackOK,TS val 1982980123 ecr 0,nop,wscale 6], length 0
15:49:22.198683 IP 46.65.32.169.38110 > 192.168.3.10.19142: Flags [S], seq 2732408244, win 0, options
  [mss 1402,sackOK,TS val 1982980123 ecr 0,nop,wscale 6], length 0
15:49:22.198893 IP 192.53.164.139.16628 > 192.168.3.10.49382: Flags [S], seq 2346537359, win 0, options
  [mss 1402,sackOK,TS val 1982980123 ecr 0,nop,wscale 6], length 0
```

CAUTION

The SYN attack is a very high packets per second (PPS) and connections per second (CPS) attack. Under load, you can expect it to exhaust link throughput from PPS before it exhausts bits per second (BPS). Be sure to monitor the PPS rate if you see this attack because if you are only monitoring BPS, it might appear that you have more available bandwidth than you actually have.

If filtering out traffic to unused TCP ports, either at the perimeter or at the server itself, a SYN Flood toward random destination ports may not be as impactful. It is certainly impactful to unprotected devices. However, remember that even with a random destination port SYN Flood, an attacker can still consume upstream resources like internet bandwidth or exhaust PPS limitations.

It is also important to note that SYN Flood protections can still be defeated in many implementations. SYN Floods are designed to exhaust a connection table with concurrent connections. It is common for small SYN Floods to defeat traditional firewalls, even with SYN Flood protections running on them.

ACK

CHARACTERISTICS

PROTOCOL: TCP

BANDWIDTH PROFILE: High BPS, Medium PPS

PACKET SIZE: Medium (566 bytes)

NOTES: 1:1 correlation of attacker ACK to target-generated RST until target is overwhelmed

THREAT RANKING: **4**

The Mirai ACK attack is a classic ACK Flood, designed to exploit the TCP stack of its target. The default parameters are to have random source port and random destination port, but to be effective, an attacker will likely set a target destination port when targeting an application.

Here are the control parameters of the attack:

FIGURE 11: MIRAI CONSOLE FOR THE ACK ATTACK

```
mirai-user@botnet# ack 192.168.3.10 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

len: Size of packet data, default is 512 bytes
rand: Randomize packet data content, default is 1 (yes)
tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random
urg: Set the URG bit in IP header, default is 0 (no)
ack: Set the ACK bit in IP header, default is 0 (no) except for ACK flood
psh: Set the PSH bit in IP header, default is 0 (no)
rst: Set the RST bit in IP header, default is 0 (no)
syn: Set the ACK bit in IP header, default is 0 (no) except for SYN flood
fin: Set the FIN bit in IP header, default is 0 (no)
seqnum: Sequence number value in TCP header, default is random
acknum: Ack number value in TCP header, default is random
source: Source IP address, 255.255.255.255 for random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

Again, note the inclusion of the six original TCP control flags (URG, ACK, PSH, RST, SYN and FIN) as attack parameters (see Figure 11). With the most basic syntax, the attack looks like this.

```
19:42:56.895246 IP 192.168.3.111.42710 > 192.168.3.10.53757: Flags [.], seq 3568308175:3568308687, ack 2537645519, win 590, length 512
19:42:56.895525 IP 192.168.3.114.19871 > 192.168.3.10.41169: Flags [.], seq 1875737928:1875738440, ack 402782860, win 26807, length 512
19:42:56.895741 IP 192.168.3.112.58193 > 192.168.3.10.49870: Flags [.], seq 1994121555:1994122067, ack 806893676, win 33842, length 512
19:42:56.896020 IP 192.168.3.115.59235 > 192.168.3.10.24840: Flags [.], seq 2782227450:2782227962, ack 597071798, win 9404, length 512
19:42:56.896234 IP 192.168.3.113.51316 > 192.168.3.10.15505: Flags [.], seq 4095646878:4095647390, ack 1047112439, win 61320, length 512
```

Note that again the target (192.168.3.10) replies with resets (RST) and should theoretically do so at a 1:1 correlation until the target is overwhelmed.

```
19:42:57.336744 IP 192.168.3.10.64016 > 192.168.3.113.41151: Flags [R], seq 1338934557, win 0, length 0
19:42:57.336977 IP 192.168.3.10.63542 > 192.168.3.112.59271: Flags [R], seq 426925092, win 0, length 0
19:42:57.337531 IP 192.168.3.10.30093 > 192.168.3.111.10362: Flags [R], seq 1517995527, win 0, length 0
19:42:57.337758 IP 192.168.3.10.29823 > 192.168.3.114.39888: Flags [R], seq 4215075265, win 0, length 0
19:42:57.338302 IP 192.168.3.10.3589 > 192.168.3.115.29099: Flags [R], seq 2643336227, win 0, length 0
```

Random source IPs do work with this attack.

```
19:46:57.595991 IP 98.141.191.54.30758 > 192.168.3.10.49263: Flags [.], seq 1042945450:1042945962, ack 4144566888, win 590, length 512
19:46:57.596308 IP 199.130.219.191.4342 > 192.168.3.10.61216: Flags [.], seq 5913103:5913615, ack 2889137304, win 61320, length 512
19:46:57.596590 IP 210.229.28.44.9331 > 192.168.3.10.10247: Flags [.], seq 1749665412:1749665924, ack 850610110, win 26807, length 512
19:46:57.596806 IP 228.4.78.132.58528 > 192.168.3.10.24469: Flags [.], seq 2111540203:2111540715, ack 174371511, win 9404, length 512
19:46:57.597025 IP 92.149.172.106.43468 > 192.168.3.10.18980: Flags [.], seq 1297054927:1297055439, ack 1118649891, win 33842, length 512
```

The result is that the attacked host will also send RSTs back to real public IPs.

```
19:46:57.593114 IP 192.168.3.10.50896 > 183.26.139.111.7712: Flags [R], seq 3686701077, win 0, length 0
19:46:57.593337 IP 192.168.3.10.58711 > 81.112.26.171.21288: Flags [R], seq 691620999, win 0, length 0
19:46:57.593612 IP 192.168.3.10.52558 > 7.22.254.114.38637: Flags [R], seq 3317558542, win 0, length 0
19:46:57.593834 IP 192.168.3.10.44742 > 144.34.26.48.16742: Flags [R], seq 1486491568, win 0, length 0
19:46:57.594057 IP 192.168.3.10.8165 > 157.90.34.180.48681: Flags [R], seq 1155229419, win 0, length 0
```

UDP

CHARACTERISTICS

PROTOCOL: UDP

BANDWIDTH PROFILE: High BPS, Moderate PPS

PACKET SIZE: Medium (554 bytes)

NOTES: Attack quickly crashes real IoT camera

THREAT RANKING: **3**

The Mirai UDP attack is unique among other UDP Floods. While still a UDP Flood, the default behavior of Mirai is to randomize the source port and the destination ports. When combined with multiple source IPs (coming from multiple bots), the result is a flood of UDP traffic that can be difficult to fingerprint on an upstream router or firewall because there is no common source IP, source port or destination port.

In the lab for this document, the UDP attack performs as expected on the Raspberry Pi bots, but when testing a true infection scenario with a real IoT device, it was observed that the cameras would crash shortly after execution of the attack. Remember that when Mirai devices crash or are rebooted, Mirai is flushed from the device. This might be why the UDPPLAIN attack exists too.

Here are the control parameters of the attack:

FIGURE 12: MIRAI CONSOLE FOR THE UDP ATTACK

```
mirai-user@botnet# udp 192.168.3.10 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

tos: TOS field value in IP header, default is 0
ident: ID field value in IP header, default is random
ttl: TTL field in IP header, default is 255
len: Size of packet data, default is 512 bytes
rand: Randomize packet data content, default is 1 (yes)
df: Set the Dont-Fragment bit in IP header, default is 0 (no)
sport: Source port, default is random
dport: Destination port, default is random
source: Source IP address, 255.255.255.255 for random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

With the most basic syntax, the UDP attack looks like this (but coming from multiple, real public IP addresses).

```
22:52:08.727350 IP 192.168.3.104.63613 > 192.168.3.10.46845: UDP, length 512
22:52:08.727392 IP 192.168.3.102.50744 > 192.168.3.10.2164: UDP, length 512
22:52:08.727438 IP 192.168.3.102.46476 > 192.168.3.10.30461: UDP, length 512
22:52:08.727485 IP 192.168.3.102.16804 > 192.168.3.10.11612: UDP, length 512
22:52:08.727531 IP 192.168.3.102.899 > 192.168.3.10.40079: UDP, length 512
```

As mentioned earlier, not all variables in the code work properly. For example, when specifying a random source IP address, the bots send a source IP of 255.0.0.0.

```
22:47:08.333547 IP 255.0.0.0.4257 > 192.168.3.10.25548: UDP, length 512
22:47:08.333594 IP 255.0.0.0.23384 > 192.168.3.10.53404: UDP, length 512
22:47:08.333641 IP 255.0.0.0.17972 > 192.168.3.10.23926: UDP, length 512
22:47:08.333686 IP 255.0.0.0.45162 > 192.168.3.10.43349: UDP, length 512
22:47:08.333732 IP 255.0.0.0.60420 > 192.168.3.10.41876: UDP, length 512
```

TIP

Remember that the packet length is adjustable. Modifying the packet length is easy during an attack, so don't rely on it as a manual filter parameter unless you must.

UDPPLAIN

CHARACTERISTICS

PROTOCOL: UDP

BANDWIDTH PROFILE: High BPS, Medium PPS

PACKET SIZE: Medium (554 bytes)

NOTES: In the wild, it could be high PPS

THREAT RANKING: 2

As the name suggests, this is a UDP attack that has less options than the other UDP attack. The description says that it's meant for higher PPS, and this is observed on some of the IoT devices. On the Raspberry Pi bots and one brand of camera, this behavior was not observed, but on another brand of IP camera, this was indeed the case.

Using that brand of IP cameras to test, the UDPPLAIN attack was approximately four times stronger than the UDP attack in both BPS and PPS.

One reason for this could be the use of a common destination port used for each particular attacking bot. By not having to randomize source and destination port, the attacking IoT device can apply more resources to the attack itself. Note that a real world attack will still have random destination ports, but the IoT devices themselves won't have to randomize them for individual attacks.

Further, it was observed that the UDP attack crashes one model of IP cameras in the lab and forces a separation from the botnet. The UDPPLAIN attack might also be a response to this behavior.

Here are the control parameters of the attack:

FIGURE 13: MIRAI CONSOLE FOR THE UDPPLAIN ATTACK

```
mirai-user@botnet# udpplain 192.168.3.10 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

len: Size of packet data, default is 512 bytes
rand: Randomize packet data content, default is 1 (yes)
dport: Destination port, default is random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet#
```

With the most basic syntax, the attack looks like this.

```
19:37:45.420785 IP 192.168.3.113.38601 > 192.168.3.10.25800: UDP, length 512
19:37:45.421015 IP 192.168.3.114.51981 > 192.168.3.10.36867: UDP, length 512
19:37:45.421603 IP 192.168.3.112.1941 > 192.168.3.10.45313: UDP, length 512
19:37:45.421823 IP 192.168.3.111.42962 > 192.168.3.10.56887: UDP, length 512
19:37:45.422033 IP 192.168.3.115.2483 > 192.168.3.10.6558: UDP, length 512
```

Remember that in random destination port UDP Floods, the server being attacked is also typically tasked with generating an ICMP unreachable message when receiving a packet on an unopened port.

```
19:38:20.671766 IP 192.168.3.10 > 192.168.3.111: ICMP 192.168.3.10 udp port 56887
  unreachable, length 548
19:38:20.672451 IP 192.168.3.10 > 192.168.3.113: ICMP 192.168.3.10 udp port 25800
  unreachable, length 548
```

ICMP messages are generated by the CPU of the device, so even if the device isn't being overwhelmed by BPS or PPS, the task of generating ICMP Unreachable messages can also impact it. There are ways to prevent this at the OS level too, so don't forget to check it.

Finally, similar to the UDP attack, a random source port will appear on an inbound attack and a random destination port will appear if the attacker hasn't modified it. This might be difficult to fingerprint and block without the right tools.

A random source port will appear on an inbound attack, and a random destination port will appear if the attacker hasn't modified it — making it difficult to fingerprint and block without the right tools.

HTTP

CHARACTERISTICS

PROTOCOL: TCP (HTTP)

BANDWIDTH PROFILE: Low BPS, Low PPS

PACKET SIZE: Medium-Small (373 bytes)

NOTES: Incredibly versatile for crafted HTTP attacks.
High amplification factor.

THREAT RANKING: 1

The HTTP attack included in Mirai is a highly flexible attack with several customizations that could prove difficult to defend against without the right tools. The most obvious attack to execute when you first encounter the tool is an HTTP GET attack, which is a traditional GET attack.

Here are the control parameters of the attack:

FIGURE 14: MIRAI CONSOLE FOR THE HTTP ATTACK

```
List of flags key=val seperated by spaces. Valid flags for this method are

domain: Domain name to attack
dport: Destination port, default is random
method: HTTP method name, default is get
postdata: POST data, default is empty/none
path: HTTP path, default is /
conns: Number of connections

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
mirai-user@botnet# http 192.168.3.10 120 domain=example.com
```

The console says that the destination port is default random, but this is inaccurate. The default destination port is TCP/80. You might notice that there are less parameters in this attack than with the others, but its implementation is quite advanced because of the type of attack it is as well as the allowed parameters in the code.

The HTTP Flood is an attack on running applications, so the victim usually cannot simply block TCP/80 traffic. The attacker has likely performed reconnaissance on the victim's network to identify the most effective attack method in the HTTP Flood attack.

A traditional GET attack is interesting on its own. In this attack, the attacker simply continues to request legitimate HTTP resources repeatedly. The requests come from real IPs that present themselves as real HTTP clients (with five legitimate HTTP user agents included in code), and they perform real HTTP requests that the attacker can customize from the Mirai console.

The premise of the attack is to exhaust resources on the target, which will continue to serve content to the requestors (attackers). The target becomes so busy serving content to the attacking bots that it ultimately cannot serve content to legitimate users, thus achieving a denial of service.

There is a very interesting amplification factor in HTTP GET attacks. In the lab, one bot generates 892 Kbps of HTTP GETs and corresponding TCP ACKs. However, the web server being attacked generates 19.6 Mbps of traffic back to that single bot. That's a 22x amplification factor with just the default Apache2/Debian "It works" page. A content-rich target will suffer from a much higher amplification factor.

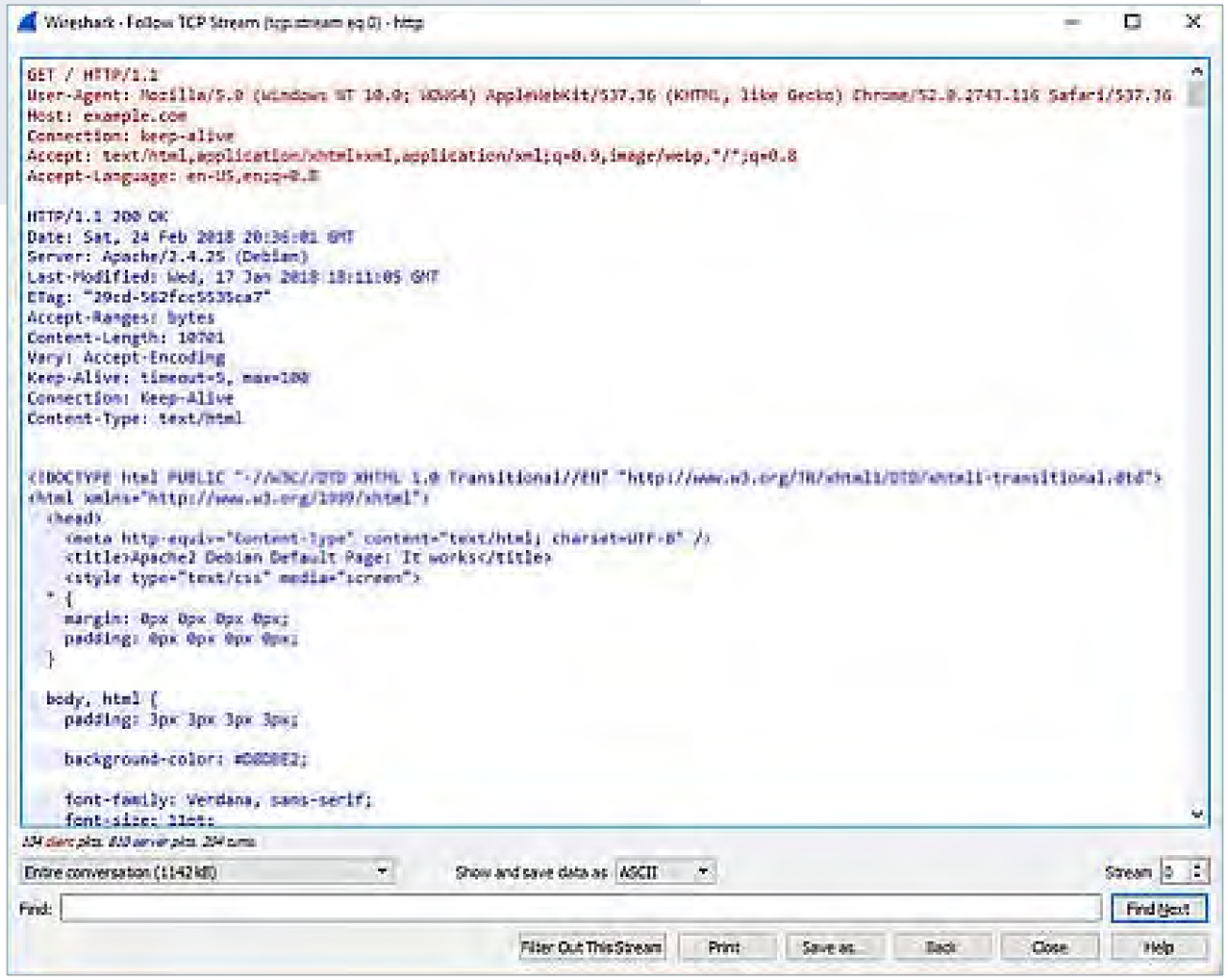
With the default syntax, the attack traffic looks like this:

```
16:36:01.971487 IP 192.168.3.111.39122 > 192.168.3.10.80: Flags [.], ack 22024, win 591, options
  [nop,nop,TS val 50329769 ecr 50327193], length 0
16:36:01.971536 IP 192.168.3.113.58454 > 192.168.3.10.80: Flags [P.], seq 1:302, ack 1, win 229, options
  [nop,nop,TS val 50329466 ecr 50327192], length 301: HTTP: GET / HTTP/1.1
16:36:01.971682 IP 192.168.3.10.80 > 192.168.3.113.58454: Flags [.], ack 302, win 235, options [nop,nop,TS
  val 50327193 ecr 50329466], length 0
16:36:01.972483 IP 192.168.3.10.80 > 192.168.3.113.58454: Flags [.], seq 1:1449, ack 302, win 235, options
  [nop,nop,TS val 50327193 ecr 50329466], length 1448: HTTP: HTTP/1.1 200 OK
16:36:01.972504 IP 192.168.3.10.80 > 192.168.3.113.58454: Flags [.], seq 1449:2897, ack 302, win 235,
  options [nop,nop,TS val 50327193 ecr 50329466], length 1448: HTTP
16:36:01.972565 IP 192.168.3.10.80 > 192.168.3.113.58454: Flags [.], seq 2897:4345, ack 302, win 235,
  options [nop,nop,TS val 50327193 ecr 50329466], length 1448: HTTP
16:36:01.972581 IP 192.168.3.10.80 > 192.168.3.113.58454: Flags [.], seq 4345:5793, ack 302, win 235,
  options [nop,nop,TS val 50327193 ecr 50329466], length 1448: HTTP
```

The target becomes so busy serving content to the attacking bots that it ultimately cannot serve content to legitimate users, thus achieving a denial of service.

Note that the traffic looks exactly like normal HTTP traffic. The following Wireshark analysis shows the decoded TCP stream of an HTTP GET attack from a Mirai bot.

FIGURE 15: WIRESHARK RENDERING OF A MIRAI HTTP GET ATTACK



Notice the HTTP user agent included in Figure 15. The default Mirai code includes five user agents, which can be presented in an attack.

```
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/51.0.2704.103 Safari/537.36
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/52.0.2743.116 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/51.0.2704.103 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/52.0.2743.116 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/601.7.7
  (KHTML, like Gecko) Version/9.1.2 Safari/601.7.7
```

So far only one type of HTTP attack has been reviewed, which is likely the most basic with the least amount of input from the user. However, it's important to understand the precision that is granted to an attacker using the tool.

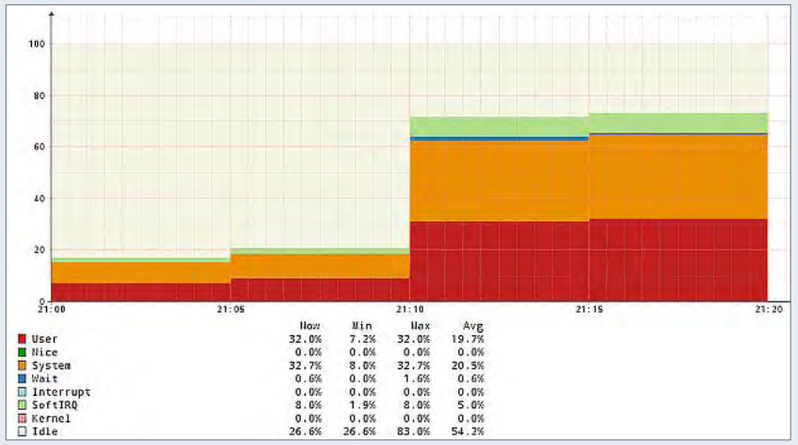
Another interesting attack in the HTTP vector is an HTTP POST, which is invoked using the "method=" parameter from the Mirai console. Although HTTP GET attacks are designed to pull static content from web servers, the HTTP POST attack is intended to abuse the use of forms or input fields on a website.

The result of the HTTP POST attack will also depend on the system being attacked, but the source will be the same — real devices emulating real HTTP clients with a custom POST that the attacker likely crafted specifically for his/her victim.

Finally, there is another interesting technique involving this attack vector. The most common HTTP Method attacks are GET and POST, but the rest of the HTTP methods can be easily exploited. The result is a CPU impact rather than network bandwidth exploitation.

For example, in an HTTP DELETE attack, the target replies to the attackers with an HTTP 405/Method Not Allowed message. The processing of this request can be CPU intensive, so although the attack is not volumetric in nature, the impact on the CPU is notable. Below you can see the CPU utilization of the target (a small server with four cores) when the HTTP Method abuse attack is launched from five lab bots.

FIGURE 16: OBSERVIUM CE GRAPH OF TARGET CPU DURING A MIRAI HTTP DELETE ATTACK



As you can see, the HTTP attack in Mirai can be used to deliver not only traditional GET attacks but also complex, precision-targeted HTTP attacks.

Defending against the HTTP attacks will prove incredibly difficult without the right tools. A NetFlow-based DDoS solution might detect an anomaly in traffic, but it won't be able to easily separate legitimate traffic from attack traffic because it's all real traffic at OSI Layers 3–4. Similarly, a router ACL or firewall policy won't be able to isolate only bad traffic.

03

Attacks Included in Mirai Variants

With a proven framework, Mirai was leveraged by attackers seeking to create their own custom variants.

For example, Masuta and DaddysMirai include the original Mirai vectors but removed the HTTP attack.

```
#define ATK_VEC_UDP 0
#define ATK_VEC_VSE 1
#define ATK_VEC_DNS 2
#define ATK_VEC_SYN 3
#define ATK_VEC_ACK 4
#define ATK_VEC_STOMP 5
#define ATK_VEC_GREIP 6
#define ATK_VEC_GREETH 7
#define ATK_VEC_UDP_PLAIN 8
```

Orion is an exact copy of the original Mirai attack table (and, just like Mirai, has abandoned the PROXY attack).

```
#define ATK_VEC_UDP           0  /* Straight up UDP flood */
#define ATK_VEC_VSE           1  /* Valve Source Engine query flood */
#define ATK_VEC_DNS           2  /* DNS water torture */
#define ATK_VEC_SYN           3  /* SYN flood with options */
#define ATK_VEC_ACK           4  /* ACK flood */
#define ATK_VEC_STOMP         5  /* ACK flood to bypass mitigation devices */
#define ATK_VEC_GREIP         6  /* GRE IP flood */
#define ATK_VEC_GREETH        7  /* GRE Ethernet flood */
//#define ATK_VEC_PROXY       8  /* Proxy knockback connection */
#define ATK_VEC_UDP_PLAIN     9  /* Plain UDP flood optimized for speed */
#define ATK_VEC_HTTP      10 /* HTTP layer 7 flood */
```

Owari added two new vectors, STD and XMAS.

```
#define ATK_VEC_UDP           0
#define ATK_VEC_VSE           1
#define ATK_VEC_DNS           2
#define ATK_VEC_SYN           3
#define ATK_VEC_ACK           4
#define ATK_VEC_STOMP         5
#define ATK_VEC_GREIP         6
#define ATK_VEC_GREETH        7
#define ATK_VEC_UDP_PLAIN  8
#define ATK_VEC_STD           9
#define ATK_VEC_XMAS      10
```

STD

CHARACTERISTICS

PROTOCOL: UDP

BANDWIDTH PROFILE: High BPS, Medium PPS

PACKET SIZE: Large (1,024 bytes)

NOTES: Nearly identical to Mirai UDPPLAIN but larger payload

THREAT RANKING: **NOT CALCULATED**

SAME PROFILE AS MIRAI'S UDPPLAIN IF PACKET LENGTH IS SET

The STD attack is a new attack vector that appears in Owari. It is a replica of the Mirai UDPPLAIN attack except that it defaults to a packet length of 1,024 bytes instead of 512 bytes.

In the Owari variant tested for this research, the console appears as follows:

FIGURE 17: OWARI CONSOLE FOR THE STD ATTACK

```
~> std 192.168.3.10 10 ?
List of flags key=val seperated by spaces. Valid flags for this method are

len: Size of packet data, default is 512 bytes
sport: Source port, default is random
dport: Destination port, default is random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
~>
```

The console states that the default packet length is 512 bytes, but this is incorrect. The default packet length is 1,024 bytes.

Also like Mirai's UDPPLAIN attack, the source and destination ports are random per flow. They are not random per packet like a UDP attack. On the wire, the attack looks like this when launched with the most basic syntax:

```
04:19:03.092763 IP 192.168.3.101.41615 > 192.168.3.10.38509: UDP, length 1024
04:19:03.093225 IP 192.168.3.100.63537 > 192.168.3.10.8420: UDP, length 1024
04:19:03.093674 IP 192.168.3.101.41615 > 192.168.3.10.38509: UDP, length 1024
04:19:03.094123 IP 192.168.3.100.63537 > 192.168.3.10.8420: UDP, length 1024
04:19:03.094573 IP 192.168.3.101.41615 > 192.168.3.10.38509: UDP, length 1024
04:19:03.095023 IP 192.168.3.100.63537 > 192.168.3.10.8420: UDP, length 1024
```

This attack is not given a threat ranking score because it was run on IP cameras instead of Raspberry Pis, for reasons described earlier. However, the relevant code of this attack matches the Mirai UDPPLAIN attack, so if the packet length is set to 1,024 bytes upon execution, the behavior will be the same.

The console states that the default packet length is 512 bytes, but this is incorrect. The default packet length is 1,024 bytes.

XMAS

CHARACTERISTICS

PROTOCOL: TCP

BANDWIDTH PROFILE: High BPS, Low PPS

PACKET SIZE: 822 BYTES

NOTES: Sets invalid TCP flag combinations, creating malformed packets

THREAT RANKING: **NOT CALCULATED**

SIMILAR THROUGHPUT PROFILE AS MIRAI'S STOMP ATTACK

The XMAS attack is the other new attack that appears in the Owari variant of Mirai, and it is interesting for several reasons. Most importantly, every sample of code that Radware observed in the wild had an error for this attack.

The attack that was launched from the bots was not the XMAS attack, it was the STD attack. This is a simple error by its author, but the error has propagated into other public variants of Owari. Fixing the mistake produces the correct attack.

The XMAS attack is a Christmas tree attack. In a Christmas tree attack, many different TCP flags are enabled, and the etymology of the name is that the packet is illuminated like a Christmas tree. In the case of Owari, every TCP flag except ECN-nonce is set.

FIGURE 18: WIRESHARK RENDERING OF TCP FLAGS SET IN THE OWARI XMAS ATTACK

```
Flags: 0x8ff (FIN, SYN, RST, PSH, ACK, URG, ECN, CWR, Reserved)
    100. .... .... = Reserved: Set
    ...0 .... .... = Nonce: Not set
    .... 1... .... = Congestion Window Reduced (CWR): Set
    .... .1.. .... = ECN-Echo: Set
    .... ..1. .... = Urgent: Set
    .... ...1 .... = Acknowledgment: Set
    .... .... 1... = Push: Set
  > .... .... .1.. = Reset: Set
  > .... .... ..1. = Syn: Set
  > .... .... ...1 = Fin: Set
    [TCP Flags: R···CEUAPRSF]
```

The attack is also an in-session attack, meaning that the three-way handshake is established before the XMAS Flood begins. The flood will not begin unless the three-way handshake is established, so the attacker will likely target an open TCP port on its victim.

FIGURE 19: WIRESHARK RENDERING OF THE OWARI XMAS ATTACK SETUP

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.3.101	192.168.3.10	TCP	74	41289 → 80 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=4228545 TSecr=0 WS=4
2	0.000482	192.168.3.10	192.168.3.101	TCP	74	80 → 41289 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=170240211
3	0.001503	192.168.3.101	192.168.3.10	TCP	66	41289 → 80 [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=4228546 TSecr=170240211
4	0.006902	192.168.3.101	192.168.3.10	TCP	822	65023 → 80 [FIN, SYN, RST, PSH, ACK, URG, ECN, CWR, Reserved] Seq=0 Ack=1 Win=32263 Urg
5	0.007031	192.168.3.101	192.168.3.10	TCP	822	[TCP Port numbers reused] 65023 → 80 [FIN, SYN, RST, PSH, ACK, URG, ECN, CWR, Reserved]
6	0.007140	192.168.3.101	192.168.3.10	TCP	822	[TCP Port numbers reused] 65023 → 80 [FIN, SYN, RST, PSH, ACK, URG, ECN, CWR, Reserved]
7	0.007242	192.168.3.101	192.168.3.10	TCP	822	[TCP Port numbers reused] 65023 → 80 [FIN, SYN, RST, PSH, ACK, URG, ECN, CWR, Reserved]
8	0.007340	192.168.3.101	192.168.3.10	TCP	822	[TCP Port numbers reused] 65023 → 80 [FIN, SYN, RST, PSH, ACK, URG, ECN, CWR, Reserved]
9	0.007438	192.168.3.101	192.168.3.10	TCP	822	[TCP Port numbers reused] 65023 → 80 [FIN, SYN, RST, PSH, ACK, URG, ECN, CWR, Reserved]

In the Owari variant tested for this research, the console appears as follows:

FIGURE 20: OWARI CONSOLE FOR THE XMAS ATTACK

```
-> xmas 192.168.3.10 120 ?
List of flags key=val seperated by spaces. Valid flags for this method are

len: Size of packet data, default is 512 bytes
sport: Source port, default is random
dport: Destination port, default is random

Value of 65535 for a flag denotes random (for ports, etc)
Ex: seq=0
Ex: sport=0 dport=65535
->
```

The flood will be volumetric in nature, but the payload of the flood is a malformed packet. There are techniques to mitigate this problematic traffic, such as blocking packet anomalies, but remember that an attack will likely also be volumetric in nature, and it can saturate a network.

This attack is not given a threat ranking score because it was run on IP cameras instead of Raspberry Pis, for reasons described earlier.

04

Burst Attacks

Burst attacks are a unique challenge for the victim to detect and mitigate without the right tools.

The attack is more of an implementation or technique than a vector and as such, can be used with many types of attack vectors. In fact, this is one of its strength because changing something about the attack in each wave makes it even more difficult to defend against.

IoT botnets like Mirai have made burst attacks more prevalent today than in the past. In the case of Mirai, the attacker has precision control — down to the second — of what an attack will look like. Each burst or wave can be configured to be slightly different, causing chaos for the victim who is not properly protected.

For example, let's say that the victim has received a 5 Gbps burst of a UDP Flood toward his/her DNS server that lasted 30 seconds. Let's assume that it was detected on the first burst and perhaps was identified as a UDP Flood toward destination port 53 with a packet length of 512 bytes. With a manual mitigation technique, an engineer must create a filter to block this exact attack.

```
15:00:15.265411 IP 192.168.3.111.29101 > 192.168.3.10.53: UDP, length 512
15:00:15.265869 IP 192.168.3.112.11428 > 192.168.3.10.53: UDP, length 512
15:00:15.266324 IP 192.168.3.115.46810 > 192.168.3.10.53: UDP, length 512
15:00:15.266777 IP 192.168.3.114.37522 > 192.168.3.10.53: UDP, length 512
15:00:15.267304 IP 192.168.3.113.34235 > 192.168.3.10.53: UDP, length 512
```

The danger becomes apparent when the next burst has a slightly modified payload. Maybe the attacker has changed the packet length to 520 bytes, or the destination port or another parameter. When this happens, the engineer defending against the attack must manually create a new filter to block the new attack.

```
15:09:35.898311 IP 192.168.3.111.45248 > 192.168.3.10.53: UDP, length 520
15:09:35.898766 IP 192.168.3.114.60616 > 192.168.3.10.53: UDP, length 520
15:09:35.899220 IP 192.168.3.112.64291 > 192.168.3.10.53: UDP, length 520
15:09:35.899675 IP 192.168.3.115.35316 > 192.168.3.10.53: UDP, length 520
15:09:35.900133 IP 192.168.3.113.54531 > 192.168.3.10.53: UDP, length 520
```

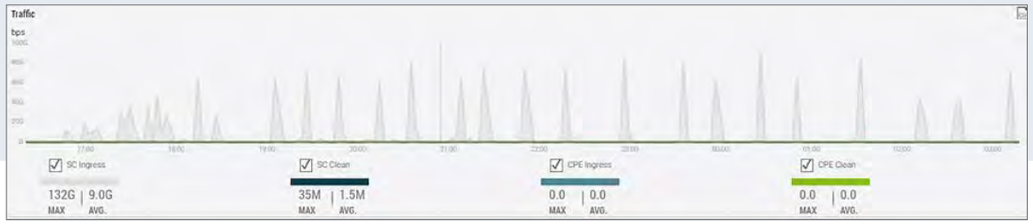
In reality, by the time the defender has manually created a new signature, the attacker is several steps ahead with new rounds of the attack. And when the attack comes from real IPs from thousands of real IoT devices, the problem of filtering is exacerbated.Now consider a more common, real-world scenario. Most networks don't collect NetFlow information or have DDoS mitigation tools on-premise. Maybe they track network utilization with SNMP graphs. The trouble is that SNMP polling is only done at predetermined intervals, usually every three to five minutes depending on the implementation.

Unless an attack lasts for the duration of two complete polling cycles, any data represented in graphs will be incorrect. You might see some of the traffic depending on its duration, but interface utilization statistics are usually the average of counters between the polling period. If that attack is a short burst on a large interface, the traffic represented in the graph will be much smaller than it actually was.

The attack will more likely manifest itself as intermittent connectivity loss, application problems or sometimes even firewall failovers if the network is fronted by high-availability firewall pairs. The application team will ask the network team what's wrong, but the network team will likely say "the network looks fine." By the time the victim realizes what's happening, the attack vector is already underway and is incredibly difficult to defend against without the right tools.

The following graph shows a recent burst attack campaign on a Radware customer spanning 10 hours.

FIGURE 21: NETWORK GRAPHS OF A REAL BURST ATTACK



With the ability to precisely control many different types of attacks, Mirai creates a challenge in defending networks without the proper tools.

05

Defense and Onward

Mirai forever changed the security threat landscape. The early attacks from the botnet were monumental, and the subsequent release of the code has inspired threat actors to push the envelope even further.

It's important to understand the capabilities of Mirai and other IoT botnets so that your organization can truly comprehend the threat.

Manually reacting to these attacks is not viable, especially in a prolonged campaign. In many cases, it is possible to block some of these attacks on infrastructure devices such as core routers or upstream transit links, but in many cases it's not. Hopefully, this handbook provides the guidance and insight needed for each vector in the event that your organization ever needs to take emergency measures.

Effectively fighting these attacks requires specialized solutions, including behavioral technologies that can identify the threats posed by Mirai and other IoT botnets. It also requires a true understanding of how to successfully mitigate the largest attacks ever seen.

Radware offers industry-leading solutions to successfully mitigate these attacks, including premise-based hardware, cloud-based services and hybrid architectures. To learn more, visit Radware.com.

Appendices

APPENDIX A: MIRAI ATTACK RATES

Mirai Attack Velocity from Five Raspberry Pi Bots				
Attack	Protocol	Packet Size	BPS	PPS
DNS	UDP	93	158,795,184	225,559
VSE	UDP	67	126,300,112	222,357
STOMP	TCP	822	488,183,616	73,877
GREETH	GRE	592	483,750,656	101,458
GREIP	GRE	578	483,389,312	103,821
SYN	TCP	74	132,103,360	211,704
ACK	TCP	566	483,049,408	105,934
UDP	UDP	554	482,700,896	108,133
UDPPLAIN	UDP	554	452,440,448	101,352
HTTP	TCP (APP)	373	3,664,168	2,277

APPENDIX B: TABLE OF FIGURES

radware

www.radware.com

ABOUT RADWARE

Radware® (NASDAQ: RDWR) is a global leader of cybersecurity and application delivery solutions for physical, cloud and software-defined data centers. Its award-winning solutions portfolio secures the digital experience by providing infrastructure, application and corporate IT protection and availability services to enterprises globally. Radware's solutions empower more than 12,500 enterprise and carrier customers worldwide to adapt quickly to market challenges, maintain business continuity and achieve maximum productivity while keeping costs down. For more information, please visit www.radware.com.

Radware encourages you to join our community and follow us on: Radware Blog, LinkedIn, Facebook, Twitter, SlideShare, YouTube, Radware Connect app for iPhone® and our security center DDoSWarriors.com that provides a comprehensive analysis of DDoS attack tools, trends and threats.