UCLouvain

École polytechnique de Louvain

Regain sight in a network blackhole

Exploitation of the Mirai botnet weaknesses

Author : **Alexis Lecocq**
Supervisors : **Alexandre Dulaunoy, Ramin Sadre**
Readers : **Jean-Michel Dricot, Alain Huet**
Academic year 2019–2020
Master [120] in Cybersecurity

1 Context

The master in cyber security is coordinated by four universities and two university colleges: “Ecole Royale Militaire”, “Université Libre de Bruxelles”, “Université Catholique de Louvain”, “Université de Namur”, “Haute Ecole de Bruxelles” and “Haute Ecole Libre de Bruxelles”. Most of the courses are given in Brussels, Belgium. You can find more information on the dedicated website[1].

The Computer Incident Response Center Luxembourg (CIRCL) is a government-driven initiative designed to provide a systematic response facility to computer security threats and incidents. CIRCL is the Community Emergency Response Team or CERT for the private sector, communes and non-governmental entities in Luxembourg[2]. Thanks to the D4 Project[3], they have access to a large amount of network packet captures, some of them coming from black holes.

To obtain the grade of master in cybersecurity, I got the opportunity to make a master thesis on an innovative cybersecurity topic. When I was looking for a subject, I met with Alexandre Dulaunoy, member of the CIRCL, who kindly accepted to help me to identify a relevant subject and gave me his support. He showed me, as an interesting starting point, a previous master thesis talking about IoT, black holes and the Mirai botnet [Mokaddem2017]. Thanks to Alexandre and the CIRCL, I had access to infrastructure and real world data to fulfill my master thesis.

Therefore, I started working at the CIRCL on February 2020. Unfortunately, one month later, the Covid-19 pandemic was spreading so fast that the office has been forced to close. I then spent the remaining time working from home, communicating through the internet. Working remotely was less efficient for me but I was well supported by Alexandre, my supervisor and my family at home. I thank them for that.

[1] https://masterincybersecurity.ulb.ac.be/ visited on 2020-02-28
[2] https://www.circl.lu/mission/ visited on 2020-02-28
[3] https://www.d4-project.org/ visited on 2020-02-28

2 Introduction

The increasing number of devices connected to the internet caused the exhaustion of IPv4 addresses. Now, more and more providers are advising a switch to IPv6 to continue deploying even more devices. All those devices are called the internet of things.

Thanks to miniaturisation, nowadays not only big physical computers are connected but small devices as well. Some of them can be industrial sensors or surveillance cameras. A lot of brands are selling those devices and the competition is huge. To make products more attractive, the industry is trying to reduce the costs. As a result, a lot of devices are often manufactured with old softwares containing unpatched security vulnerabilities. Even when a device is shipped with an up-to-date software, security vulnerabilities can be discovered during its lifetime. It is important for the owner of the device to be able to update the software quickly when a vulnerability is found. Unfortunately, due to these cost constraints, updates are often not available.

Consequently, a lot of vulnerable devices are now connected to the internet. They are let open to anyone having the technical ability to exploit them. When there were only a few of those devices, they were not an interesting target. Indeed developing an exploit has a cost and if the exploitation of devices makes less money than developing the exploit itself, doing so will not attract a lot of attackers. This explains why, at first, the importance of those vulnerable devices were underestimated. Though, technology evolves with time and devices become cheaper and affordable to an increasing number of users. More and more devices were connected and it became interesting for attackers to exploit them.

One of the most widespread ways to make money from infected devices is to run DDoS attacks from them. Even if the network bandwidth of a device is not high, a million devices together can generate quite a high amount of bandwidth. This amount of bandwidth targeting a single server can easily overwhelm it and make it unavailable to everyone, including the legitimate users. Targeting the critical infrastructure of a company with such a technique can aim to blackmail them and convince them to pay a ransom to stop the attack.

Another illegal way to make money, if the infected device has access to the corporate network, is to encrypt data stored on file servers in the corporate network. The key is sent to the attacker over the network and deleted from the infected device once the encryption is completed . The attacker then blackmails the company to pay a ransom in exchange for the key allowing them to decrypt their data. If the encryption algorithm is strong and the attacker did not make an implementation flaw, it is not possible for the company to decrypt the data without the specific key.

All those attacking scenari have increased the attractiveness of such attacks and many attackers started to make a business out of it. The creation of exploitation softwares called botnet gained popularity. Mirai is one of them and used pretty clever ways to spread more quickly than any competitor.

The main goal of the project is to discover the current status of Mirai-like botnets and to check if this botnet, and its forks, still represent an actual threat these days. It can be useful for security professionals to better defend their infrastructure. 42 Days of traffic data collected inside a network black hole will be used to make the analysis.

As Mirai uses a stateless TCP scanner, the second goal is to document the actual TCP stateless scanning techniques, comment on the possible detection and suggest improvements. This can be useful for security professionals monitoring network scanning, providing them with some hints about new network scanning techniques that could emerge and render the network monitoring less efficient in the long-term.

The third goal is to find out some ways to stop or at least mitigate the spreading of this particular botnet. The Mirai source code will be analysed to find out potential weak points. A virtual environment will be set up to test the exploitation of those weak points. This section can be useful to reduce the impact of the botnet and make a safer internet. Software developers may be interested in this section to improve their code quality and avoid similar mistakes.

3 Background

3.1 Distributed DoS

A distributed denial-of-service[4] or DDoS attack is a malicious attempt to disrupt normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. DDoS attacks achieve effectiveness by using multiple compromised computer systems as sources of attack traffic. Exploited machines can include computers and other networked resources such as IoT devices. From a high level, a DDoS attack is like a traffic jam clogging up a highway, preventing regular traffic from arriving at its desired destination.

3.2 Botnet

A botnet[5] refers to a group of computers which have been infected by malware and have come under the control of a malicious actor. The term botnet is a contraction of the words “robot” and “network”. Each infected device is called a bot. Botnets can be designed to accomplish illegal or malicious tasks including sending spam, stealing data, encrypting data, clicking on fraudulent ads or making DDoS attacks.

3.3 Network telescope or blackhole

A network telescope[6] (also called a black hole, an Internet sink, darkspace, or a darknet) is a globally routed network that carries almost no legitimate traffic because there are few provider-allocated IP addresses in this prefix. After discarding the legitimate traffic from the incoming packets, the remaining data represent a continuous view of anomalous unsolicited traffic called Internet Background Radiation or IBR. IBR results from a wide range of events, such as backscatter from randomly spoofed source denial-of-service attacks, the automated spread of Internet worms and viruses, scanning of address space by attackers or malware

[4] https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ visited on 2020-02-25
[5] https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/ visited on 2020-02-25
[6] https://www.caida.org/projects/network_telescope/ visited on 2020-08-12

looking for vulnerable targets, and various misconfigurations (e.g. mistyping an IP address). In recent years, traffic destined to darkspace has evolved to include longer-duration, low-intensity events intended to establish and maintain botnets.

3.4 D4 Project

D4 Project[7] is a large-scale distributed sensor network to monitor DDoS and other malicious activities relying on an open and collaborative project. It is maintained and hosted by the CIRCL. This project consists of a central core D4 Server collecting data. D4 clients, installed on all kinds of sensors, send data to this server using the D4 Protocol. This is a minimal protocol that can be used to transfer packet captures, JSON data, log lines, DNS records and more[8]. Server modules can be installed to consume the data stored by the D4 Server. An example of such module is the Passive DNS one[9]. This module parses DNS data and inserts them into a Passive DNS server, which can be queried later.

3.5 TCP handshake

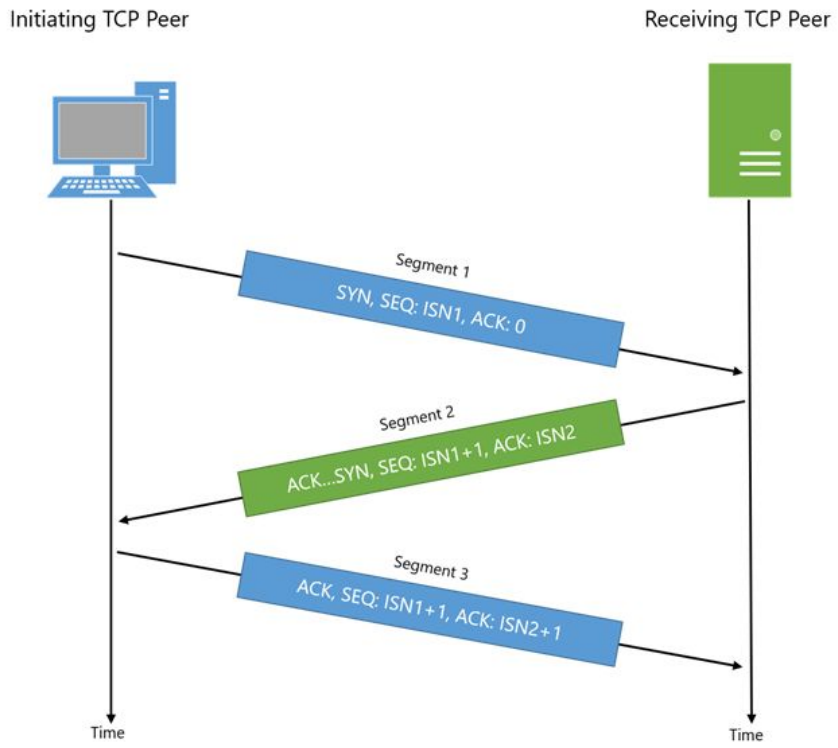
TCP handshake is the initial TCP connection establishment. It happens before any data can be sent. Here's the sequence from the TCP RFC 793[10]:

1) A --> B SYN my sequence number is X
2) A <-- B ACK your sequence number is X
3) A <-- B SYN my sequence number is Y
4) A --> B ACK your sequence number is Y

Because step flags ACK and SYN present at step 2) and 3) can be sent inside a single packet, we call this a 3-way handshake. The sequence number sent from A to B is called Initial Sequence Number 1 or ISN1. The sequence number sent from B to A is called the ISN2.

[7] https://www.d4-project.org/ visited on 2020-02-24
[8] https://github.com/D4-project/architecture/tree/master/format visited on 2020-02-24
[9] https://github.com/D4-project/analyzer-d4-passivedns visited on 2020-02-24
[10] https://tools.ietf.org/html/rfc793#page-27

Here is a clear schema from johnpfernandes.com[11]:



3.6 Broadband Forum

Broadband Forum is an open, non-profit industry organization composed of the industry's leading broadband operators, vendors, and thought leaders who are shaping the future of broadband. Their work to date has been the foundation for broadband's global proliferation and innovation. For example, the Forum's flagship TR-069 CPE WAN Management Protocol has nearly 1 billion installations worldwide.[12]

[11] https://www.johnpfernandes.com/2018/12/08/the-tcp-3-way-handshake/ visited on 2020-03-06
[12] https://www.broadband-forum.org/about-bbf visited on 2020-03-05

4 State of the art

4.1 Internet of things

The subject of weak security affecting internet devices has been largely covered in previous papers. Those devices have many restrictions and limitations in terms of the components and devices, computational and power resources [Mahmoud2015]. Those constraints often makes it impossible to run common secure software implementation. In the heterogeneous IoT, diversified hardware platforms and customized operating systems make it difficult to educate programmers on security awareness[Zhang2014]. Such diversity does not make it an easy task to develop universal secure softwares. A number of research works identified that IoT devices have vulnerabilities exposed to attackers [Zhang2014].

The main goal of manufacturers is to make profit. Securing IoT is not easy and has a very high cost. Does this security affect the sale? The end user usually has little technical knowledge and hardly knows if a device is really secured. He can only rely on the information he receives from others and unfortunately, the seller's marketing is a big part of it. Indeed, it is more efficient to advertise security than securing devices themself [Lo2018].

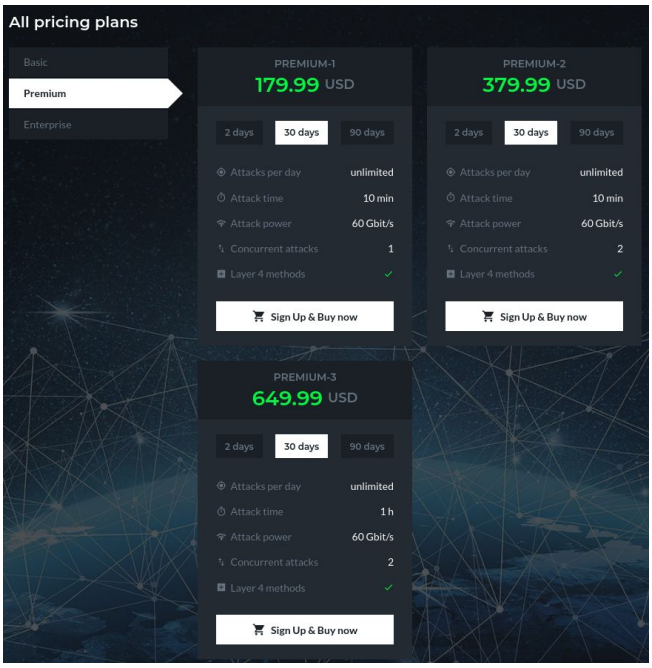
4.2 Mirai

A few years ago, the Mirai botnet raised a lot of attention because it caused a few major DDoS attacks around August 2016[13]. The author later released the source code on hackforums under the name Anna-senpai[14]. The source code of it is now available on GitHub[15]. Thanks to this release, we can now read and study the botnet code to find out his mysteries.

Previous studies mainly focus on Mirai's targets, infection procedure and the goal of the botnet. Mirai targets internet devices with a low level of security. Mirai uses default credentials to gain access to those devices. Back in 2016, Mirai's primary goal was mainly to execute DDoS

[13] https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/. Visited on 25/02/2020.
[14] https://hackforums.net/showthread.php?tid=5420472. Visited on 25/02/2020.
[15] https://github.com/jgamblin/Mirai-Source-Code. Visited on 25/02/2020.

attacks. Those attacks can be sold as “booters” or “stressers” or just DDoS on the black market. The price depends on the bandwidth capacity and the attack duration. Here is an example from StressThem[16], a DDoS provider accessible as a Tor hidden service:



4.2.1 Stateless port scan

Mirai author had very good ideas for his botnet to spread faster than any competitor. The first one was to use a stateless port scanner. No state is saved and consequently, less memory is used. This allows even hardware with low specifications and few available memory, such as an IoT device, to scan a large number of IP addresses.

According to the TCP RFC 793[17], when a client initiates a connection to a server, it should use an ISN generator *which selects a new 32 bit ISN*. The generated number is then saved on the client side, waiting for the response from the server. The Mirai ISN generator is simpler:

```
225                     tcph->seq = iph->daddr;
```

[16] https://www.stressthem.to/#pricing visited on 2020-06-09.
[17] https://tools.ietf.org/html/rfc793#page-27

The ISN1 is set to the destination IP. This way, the scanner does not have to save the ISN1 into memory and becomes stateless. As pointed out by [Wagner2017] and [Antonakakis2017], this behavior also allows easy fingerprinting and detection of Mirai port scanning. Indeed, a simple comparison between the destination IP address and the ISN1 is needed.

Mirai was targeting telnet enabled devices, scanning for opened port 23 and 2323, picking port 23 more often because it is more common (9 over 10 times):

```
221             if (i % 10 == 0)
222             {
223                 tcph->dest = htons(2323);
224             }
225             else
226             {
227                 tcph->dest = htons(23);
228             }
```

The target IP address is selected randomly. Mirai is not scanning the whole 2^{32} IPv4 address range. Indeed, a lot of addresses are avoided by the scanner: the local ones, the private ones, the multicast ones, some whitelisted ones... When scanning, Mirai picks an IPv4 address randomly among less than 3 422 552 064.

When parsing a received raw TCP packet, Mirai checks that the port is either 23 or 2323, that both flags SYN+ACK are present and that the ACKed sequence number is right:

```
270             if (htonl(ntohl(tcph->ack_seq) - 1) != iph->saddr)
271                 continue;
```

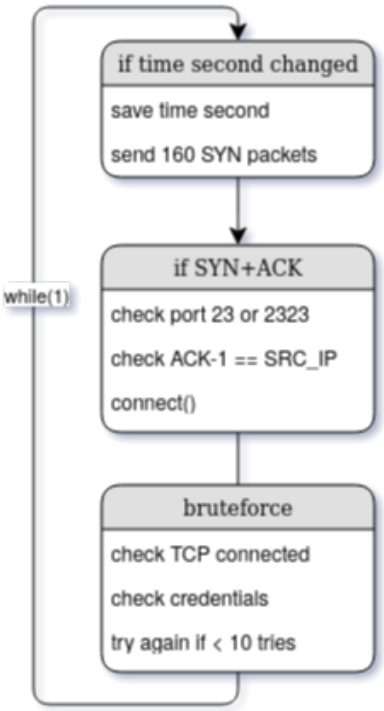
4.2.2 Credentials brute forcing

Here is another clever idea implemented by Mirai's author. The botnet contains a relatively small set of login and password combinaisons, the most widely used ones by default. When a target device is found, a bruteforce attack is started. A random set of credentials is picked and sent to the telnet server. If the botnet does not get a valid answer within 5 seconds, it will try another set. This happens 10 times in a row. If the credentials are found, they are sent to a report server

for the target to be infected and the target is forgotten. If Mirai made 10 login attempts, the target is forgotten. This may avoid locking the target device and drawing too much attention.

Because a lot of devices are running the botnet at the same time, if the valid credentials are not found during the first attack, it is likely that another client will find the same target and try other credentials. After a few attacks, there is a high chance that the whole set of credentials have been tested and that the target is effectively infected, if vulnerable.

4.2.3 Scanner state diagram



As it is visible on the left figure, both brute forcing and scanning are made in the same process. The scanning sends 160 SYN packets only if the actual time in seconds changed. In practice, that means it can send at most 160 packets per second. But if the code does not reach the SYN sender during one whole second, the SYN packets are not sent and the average packets sent per second becomes lower. The more time the program spends at receiving SYN+ACK responses and brute forcing, the less often it will reach the SYN sender, the less SYN packets will be sent and the less potential targets will be accessed.

4.3 Stateless TCP scan

Mirai is not the first stateless TCP scanner. A lot of other scanners achieve the same goal by using different techniques. I have analysed the most popular ones to identify, comment on and document their techniques.

4.3.1 Hping

Hping is a software made to easily generate network packets from string or binary representation. Hping2 from v1.33 released on 2004-03-10[18] allows the user to make port scan with the **--scan** option. The feature has been kept in hping3, present in most of Linux package repositories today. This port scanner uses a very simple technique, the ISN1 is generated randomly if not defined in arguments as we can see in the **sendtcp.c** file:

```
59  →/* sequence number and ack are random if not set */
60  →tcp->th_seq = (set_seqnum) ? htonl(tcp_seqnum) : htonl(rand());
61  →tcp->th_ack = (set_ack) ? htonl(tcp_ack) : htonl(rand());
62
63  →tcp->th_off = src_thoff + (tcp_opt_size >> 2);
64  →tcp->th_win = htons(src_winsize);
65  →tcp->th_flags = tcp_th_flags;
```

As we can see in the **main.c** file, rand is seeded by time:

```
283  →srand(time(NULL));
284
```

Because random numbers in C only depend on the seed and the number of times the function **rand** has been called, it is possible to compute a table with all times linked to their first random numbers. To detect hping port scanning, we could compare the ISN number with the random numbers in the table. This table could be computed dynamically when a packet is received, using a range of times from a few seconds ago to the next few seconds (to correct an eventual time difference between the sender and the receiver). The larger the time period is computed, the higher is the chance to detect the source scanner but the slower is the computation.

After sending TCP SYN packets, hping just displays all SYN+ACK packets without any check on the ACKed sequence number. This technique can be used because the software is running only for a short period of time (the time of the scan). On the contrary, the Mirai port scanner is running indefinitely.

[18] see CHANGES file http://www.hping.org/hping2.0.0-rc3.tar.gz visited on 2020-06-15

4.3.2 UnicornScan

Unicornscan is an information gathering and correlation engine built for and by members of the security research and testing communities. The open source UnicornScan software 0.4.2 was released by Dyad Security on 2004-09-30 on their website www.dyadsecurity.com[19] and on 2004-10-05 on SourceForge[20]. Interestingly enough, they also released a web GUI called Alicorn on the same day but this GUI has been removed later and was not available any more on 2004-12-07[21]. Unfortunately, the website www.dyadsecurity.com went offline and a new owner bought it on 2017-03-27[22] so we can only find this information thanks to internet archive[23].

The UnicornScan also uses a stateless scan. Here's the ISN1 creation code:

```
735     else if (s->ss->mode == MODE_TCPSCAN) {
736         uint32_t seq=0;
737
738         /*****************************************************************
739          *   BUILD TCP HEADER   *
740          *****************************************************************/
741         seq=(s->ss->syn_key ^ (s->ss->current_dst ^ (rport + sl.local_port)));
742
743         if (s->ss->tcpoptions_len) {
744             sl.tcpo=libnet_build_tcp_options(s->ss->tcpoptions, s->ss->tcpoptions_len, sl.libnet_h, sl.tcpo);
745             if (sl.tcpo == -1) {
746                 MSG(M_ERR, "tcpoptions: `%s'", libnet_geterror(sl.libnet_h));
747                 terminate(TERM_ERROR);
748             }
749         }
750
751         sl.tcp=libnet_build_tcp((uint16_t)sl.local_port, rport, seq, 0, s->ss->tcphdrflgs, s->ss->window_size, chksum, 0,
752         LIBNET_TCP_H + s->ss->tcpoptions_len, NULL, 0, sl.libnet_h, sl.tcp);
753         if (sl.tcp == -1) {
754             MSG(M_ERR, "tcphdr: `%s'", libnet_geterror(sl.libnet_h));
755             terminate(TERM_ERROR);
756         }
757     }
```

As we can see in file **send_packet.c** on line 741, destination IP address is also used here but it is combined with a SYN_KEY, destination port and source port.

[19] https://web.archive.org/web/20041009221405/http://www.dyadsecurity.com/s_unicornscan.html visited on 2020-06-15
[20] https://sourceforge.net/projects/osace/files/unicornscan/ visited on 2020-06-09
[21] https://web.archive.org/web/20041207034549/http://www.dyadsecurity.com:80/s_unicornscan.html visited on 2020-06-15
[22] https://whois.domaintools.com/dyadsecurity.com visited on 2020-06-09
[23] https://archive.org/web/ visited on 2020-06-15

The SYN_KEY is created randomly on software startup as we can see in the **master.c** file on line 129:

```
129     if (s->ss->mode == MODE_TCPSCAN) s->ss->syn_key=arc4random();
```

According to the documentation, the **arc4random**[24] function provides a cryptographic pseudo-random number generator. That means it is not possible to guess which value will derive from this function. Here's the TCP scanner packet parser:

```
463     sport=ntohs(t_u.t->source);
464     dport=ntohs(t_u.t->dest);
465     seq=ntohl(t_u.t->seq);
466     ackseq=ntohl(t_u.t->ack_seq);
467     doff=t_u.t->doff; res1=t_u.t->res1;
468     window=ntohs(t_u.t->window);
469     chksum=ntohs(t_u.t->check);
470     urgptr=ntohs(t_u.t->urg_ptr);
471
472     if (pk_layer == 2) {
473         uint32_t eackseq=0;
474         int res=0;
475
476         eackseq=(s->ss->syn_key ^ (r_u.i.host_addr ^ (sport + dport)));
477
478         res=ackseq - eackseq;
479
480         switch (res) {
481             case 0:
482                 MSG(M_DBG2, "hrmm what?");
483                 break;
484             case 1:
485                 if (s->verbose > 6) MSG(M_DBG1, "cool, thats right");
486                 break;
487             case 2:
488                 MSG(M_DBG2, "must be a connection?");
489                 break;
490             default:
491                 if (s->verbose > 5) MSG(M_DBG2, "Not my packet ackseq %.08x expecting somewhere around %.08x", ackseq, eackseq);
492                 return;
493                 break;
494         }
495     }
496
```

As we can see in file **packet_parse.c** on line 476, the previously computed ISN1 is computed again for the newly received packet. On line 478, the ISN1 is then subtracted from ASKed number. If the result is 1, it means the packet is a response to a previously crafted packet.

This stateless scan is a bit cleverer than the Mirai one because ISN1 is also different for each software instance, source port and destination port. As the ISN1 depends on more parameters, it is way harder to identify its usage inside a black hole. Though, the same syn_key is used for all

[24] https://man.openbsd.org/arc4random.3 visited on 2020-06-15

packets. To identify the scanner, we need to get at least 2 packets. We can compute the expected syn_key from the first packet and see if it matches the expected syn_key from the following packets.

4.3.3 Nkiller2

Nkiller2 was released by Fotis Chantzis aka ithilgore on Phrack ezine in 2009[25]. It is a PoC of an attack, similar to sockstress[26] exploiting a weakness in the TCP network stack. To allow maintaining a big number of TCP connections without using a big amount of memory, the TCP stack of Nkiller2 is completely stateless:

> "Attacker sends a group of SYN packets to the victim. In the sequence number field, he has encoded a magic number that stems from the cryptographic hash of { destination IP & port, source IP & port } and a secret key."

This idea is the most advanced I have seen so far. In this case, the cryptographic hash is the HMAC-SHA1 of both IPs, both ports using a secret key:

```
1051    /* Calculate a sha1 hash based on the quadruple and the skey */
1052    HMAC(EVP_sha1(), (char *)o.skey, strlen(o.skey), input, input_len,
1053        cookie, &cookie_len);
```

Even this good idea has room for improvement. The ISN1 is extracting only the first 32 bits of the output hash:

```
1057    /* Get only the first 32 bits of the sha1 hash */
1058    memcpy(&seq, &cookie, sizeof(seq));
1059    return seq;
```

As a general rule, it is better to have all the bits play a role to avoid specific attacks. This could be done by XORing all chunks of 32 bits together.

[25] http://phrack.org/issues/66/9.html#article visited on 2020-06-12
[26] https://defuse.ca/sockstress.htm visited on 2020-06-12

Another point could be improved. In this case, the secret key is fixed in the source code to "Nkiller31337" so it would be easy for anyone to compute the hash of any packet. it is still quite easily identifiable. Even if a malware used another key, it would be a matter of time for it to be known by reverse engineering the binary. If we do not have access to the key, it could be possible to brute force it but this would require a lot of power and I am not sure it would be worth it. To avoid storing data inside the binary and to get a different key on each execution, the key could be generated on the program startup.

The nice property of this technique is that two packets targeting two different ports have very different ISN1. Even if both IPs and both ports are reused inadvertently, the packet will contain the same ISN1 so it could be seen as a TCP retransmitted packet.

4.3.4 Nmap

The most popular tool used to scan ports is definitely nmap. According to the documentation[27], when the user has raw packet privileges (root on Linux), a stateless SYN scan is done. When the user does not have sufficient privileges, nmap automatically fallbacks to "Connect" scan, which is stateful because it is managed by the operating system. I have tested and I can confirm this behavior in practice. As I am studying a stateless scanner, I will focus on the stateless SYN scan method. **syn_scan** was available from nmap 1.51[28] but was stateful in this version. It became stateless with version 1.60-Beta[29] but this version has never been released. The version 2.00, released on 1998-12-12[30], also benefits from this improvement. The ISN1 is called **seq** and defined in file **scan_engine_raw.cc** on line **1316**:

```
1316        seq = seq32_encode(USI, tryno, pingseq);
```

[27] https://nmap.org/book/man-port-scanning-techniques.html#idm45983417623568 visited on 2020-06-15
[28] https://nmap.org/dist-old/nmap-1.51.tar.gz visited on 2020-06-15
[29] https://nmap.org/dist-old/nmap-1.60-Beta.tar.gz visited on 2020-06-15
[30] https://nmap.org/book/history-future.html visited on 2020-06-15

The function **seq32_encode** is defined in the same file on line **283**:

```
280  /* The try number or ping sequence number can be encoded into a TCP
     SEQ or ACK
281      field. This returns a 32-bit number which encodes both of these
         values along
282      with a simple checksum. Decoding is done by seq32_decode. */
283  static u32 seq32_encode(UltraScanInfo *USI, unsigned int trynum,
284                          unsigned int pingseq) {
285    u32 seq;
286    u16 nfo;
287
288    /* We'll let trynum and pingseq each be 8 bits. */
289    nfo = (pingseq << 8) + trynum;
290    /* Mirror the data to ensure it is reconstructed correctly. */
291    seq = (nfo << 16) + nfo;
292    /* Obfuscate it a little */
293    seq = seq ^ USI->seqmask;
294
295    return seq;
296  }
```

As we can see, **seq** is built using 16 bits from **trynum** and 8 bits from **pingseq**. **trynum** is the number of times the packet has been sent, which start at 0. **pingseq** is 0 if the packet is not a ping probe. The result is duplicated to obtain 32 bits and XORed with a **seqmask** constant, defined randomly in the file **scan_engine.cc** on line **911**:

```
911    seqmask = get_random_u32();
```

When a packet is received, it is checked that after XORing the mask, both 16 bits parts are equal. The function **seq32_decode** is defined in file **scan_engine_raw.cc** on line **300**:

```
298 /* Undoes seq32_encode. This extracts a try number and a port number from a
299    32-bit value. Returns true if the checksum is correct, false otherwise. */
300 static bool seq32_decode(const UltraScanInfo *USI, u32 seq,
301                          unsigned int *trynum, unsigned int *pingseq) {
302   if (trynum)
303     *trynum = 0;
304   if (pingseq)
305     *pingseq = 0;
306
307   /* Undo the mask xor. */
308   seq = seq ^ USI->seqmask;
309   /* Check that both sides are the same. */
310   if ((seq >> 16) != (seq & 0xFFFF))
311     return false;
312
313   if (trynum)
314     *trynum = seq & 0xFF;
315   if (pingseq)
316     *pingseq = (seq & 0xFF00) >> 8;
317
318   return true;
319 }
```

For the stateless behavior, the same **seqmask** is used for all ports and at first, both **trynum** and **pingseq** are set to 0. In this case, ISN1 is simply set to **seqmask** and is the same for all packets, making the technique very easy to detect both by a human and by a computer, if multiple ports are scanned:

```
TCP        58 42324 → 443 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 587 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 199 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 21 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 5900 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 8080 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 53 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 995 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
TCP        58 42324 → 139 [SYN] Seq=671984653 Win=1024 Len=0 MSS=1460
```

4.3.5 Summary

In the past, some malwares have already spread using stateless UDP scan [Shannon2004]. It was not very challenging because the UDP protocol is stateless by design. TCP is slightly different because it is stateful. As seen previously, some techniques can be used to avoid the scanner to keep a state. Here is a summary of them:

	ISN1 or seq	**how detectable**
Hping	c rand() init: srand(time(0))	🔴 hard if > 1 scan
UnicornScan	syn_key ^ dst_ip ^ (src_port+dst_port) init: syn_key=arc4random()	▯ easy if > 1 scan
Nkiller2	hash(dst_ip, dst_port, src_ip, src_port)	▯ easy if known key ● very hard if unknown key
Nmap	seqmask init: seqmask=get_random_u32()	▯ trivial if > 1 scan
Mirai	dst_ip	▯ trivial from 1 scan

Even though those techniques are known and used for years, Mirai is the first widely spread malware to use one. If the Mirai port scanner was a clever idea, it was not the cleverest port scanner at the time of its creation. Indeed, this scanner can be easily fingerprinted, unlike some other techniques. The reason for this choice may be the additional amount of computing power needed to make a better scanner.

I have read the code of all studied scanners to extract information about the implementation and for all of them, I did not find it easy to read. On the contrary, the code is old, with very big functions, big files and does not seem to follow modern convention. In this context, we could understand why no previous malware authors did use stateless scanners: there was no easy-to-use code available back then.

I have a suggestion of improvement, as a compromise between the CPU consumption of the scanner and the difficulty to detect it. The idea is to use the UnicornScan technique and

regularly renew the syn_key, once an hour for example. The computation of the ISN1 does not consume a lot of CPU because XOR and addition are hardware instructions in all CPUs. The renewal of the key would be negligible, because only happening once in a long time. One would need a very large amount of IP to be able to detect it if no more than one port per IP is scanned.

An important note is that TCP stateless scan requires the software to open a raw TCP socket, which itself requires extended privileges on most OS. Any TCP scanner running with limited privileges has to use the stateful TCP API from the OS. That is another reason that may discourage botnet developers from using a stateless TCP scanner.

Some other fingerprinting techniques could be used based on other fields. For example, it is possible to detect the OS with the TTL. Though, those fields are not used to save the state and could be set to whatever value is needed to make the packet stealthier.

4.4 Botnet DoS

Some research on botnet DoS has been released in the past. One [Lathrop2014] was exploiting a weakness in the code of Dirt Jumper botnet to make it less efficient when attacking an HTTP server. One of the botnet features was to make a huge number of HTTP requests to an HTTP server to make it unresponsive. With the evolution of the DDoS protection against this kind of attack by the web providers, Dirt Jumper's code included a smart DoS capability and became more and more complex. In the end, if the response from the HTTP server was lighter than 100 bytes, the botnet would wait for more bytes, making it much less efficient. This allowed web providers to serve a short redirection to another web URL. All legitimate clients were redirected to a valid web page while the botnet was stuck waiting for data.

One [Stone-Gross2010] has registered a domain name to take full control over the Mebroot malware. Indeed, to be able to contact the CC server even if one domain name got seized, the malware was creating the domain based on the date, using a custom algorithm. The researchers reverse engineered the communication protocol, created a CC server and registered the right domain name to take full control over the botnet for 10 days.

5 Our approach

As we have seen, Mirai botnet was only used to target TCP ports 2323 and 23. Though after Mirai source code was released, a lot of clones have started to appear on the internet. The first interesting thing to know is which ports are now targeted. In this section, I try to identify how Mirai evolved with time and what are its new targets.

5.1 Mirai-like scan analysis

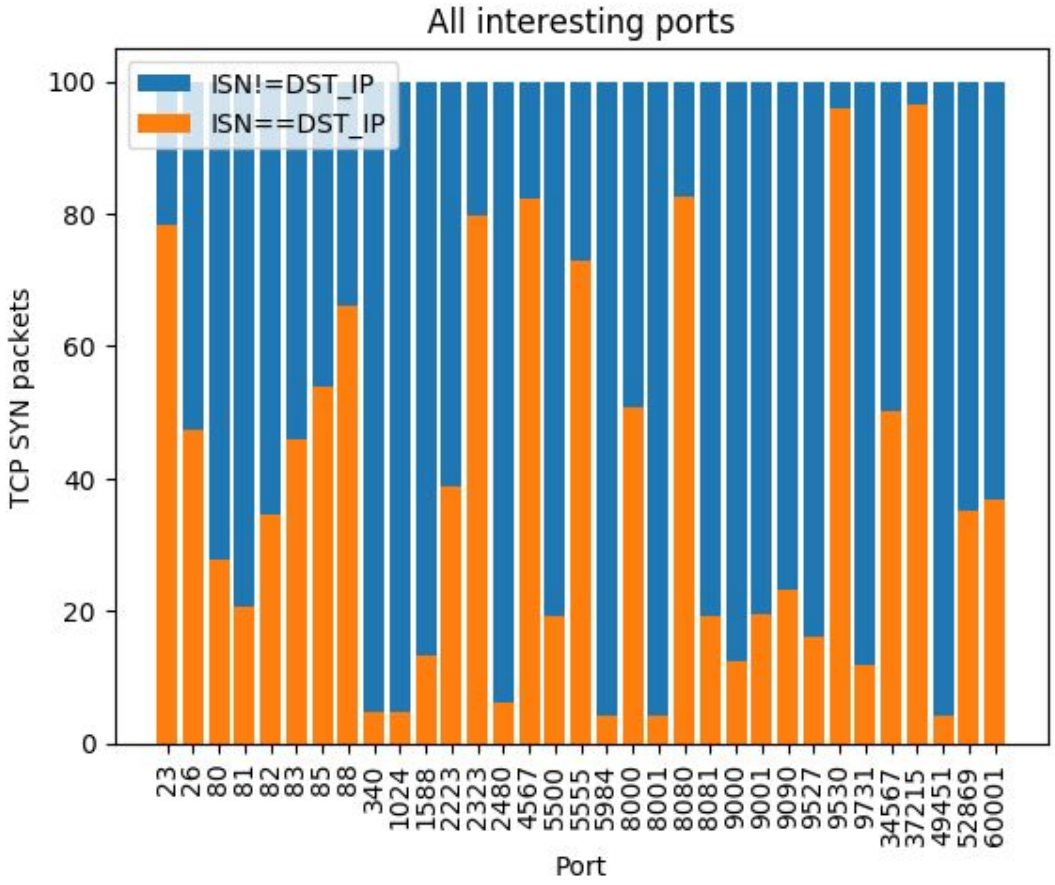
The Mirai-like port scan can be easily identified, as we have seen previously. Thanks to the CIRCL and the D4 Project, I got access to packet captures coming from a network black hole which receives data from two distinct networks: one /21 and one /22. The total number of IP addresses is about 2048+1024=3072. This black hole is a bit different from what is usually used: it replies to TCP SYN packets with TCP RST packets and to UDP packets with ICMP destination unreachable packets. As a result, it is not completely silent.

The analysed data was collected from 2020-01-16 to 2020-02-26 (both included) for a total of 42 days. During this period, a total of 251 081 607 TCP SYN packets were received. Among them, 20 495 357 packets or 8,16% contained the Mirai fingerprint.

I identified the proportion of Mirai-like SYN packets beside other scanning methods and misconfigured attempts as well as the most targeted ports by Mirai-like botnets. To avoid noise, the ports with a percentage lower than 4 or a number of SYN packets lower than 1000 have been discarded.

The probability to detect a non Mirai-like packet as one is 1 over 2^{32}=4 294 967 296 because the ISN1 has to be exactly equal to the 32bits destination IP address. Considering the number of packets received, this is very unlikely and even if a few are included, it will be negligible against the total number of packets.

Here is the resulting graphic:



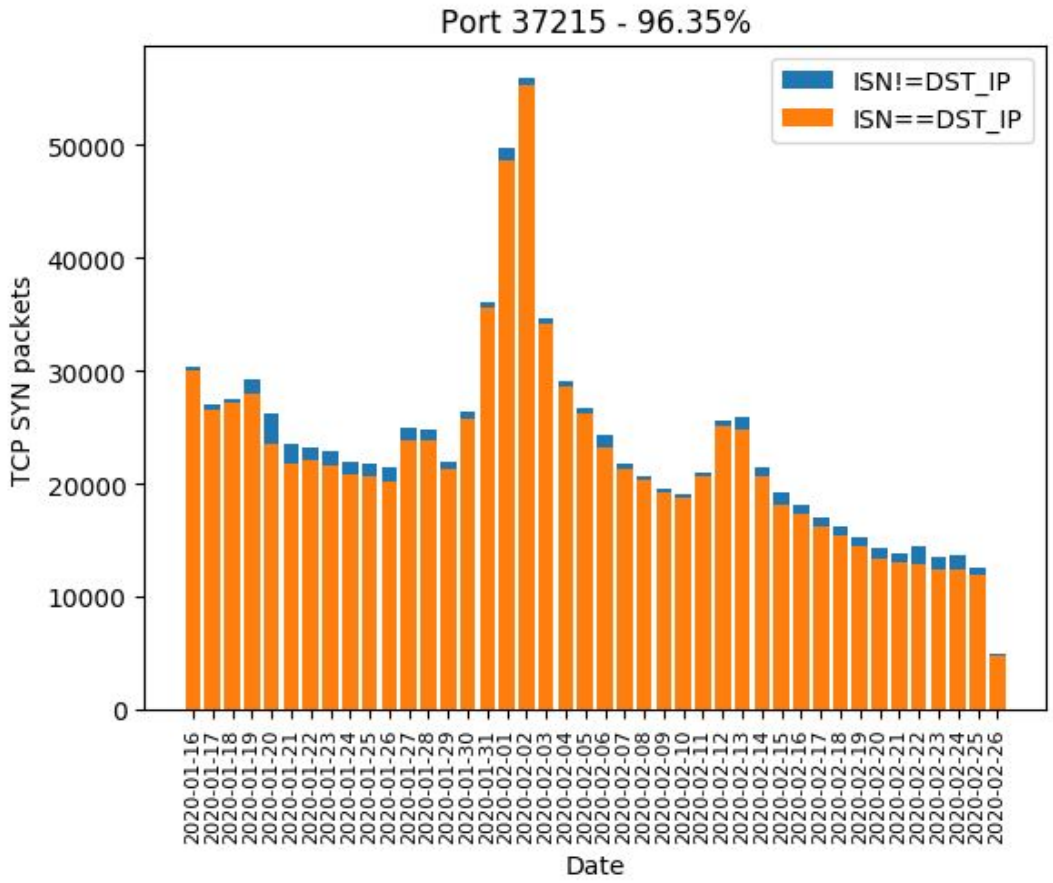
After publication of these results in a blog post[31] on the D4 Project website and announcement on Twitter[32], Bad Packets replied[33] they have seen a similar distribution of targeted ports and services. This confirms the data collected through our black hole is consistent with others collection data points.

For the most interesting ports, I have made graphics to visualize the repartition over time and I investigated to know why those particular ports were targeted.

[31] https://www.d4-project.org/2020/03/06/analyzer-d4-isn.html visited on 2020-03-06
[32] https://twitter.com/d4_project/status/1235944665419046920 visited on 2020-03-06
[33] https://twitter.com/bad_packets/status/1236082830045667330 visited on 2020-03-07

5.1.1 Port 37215

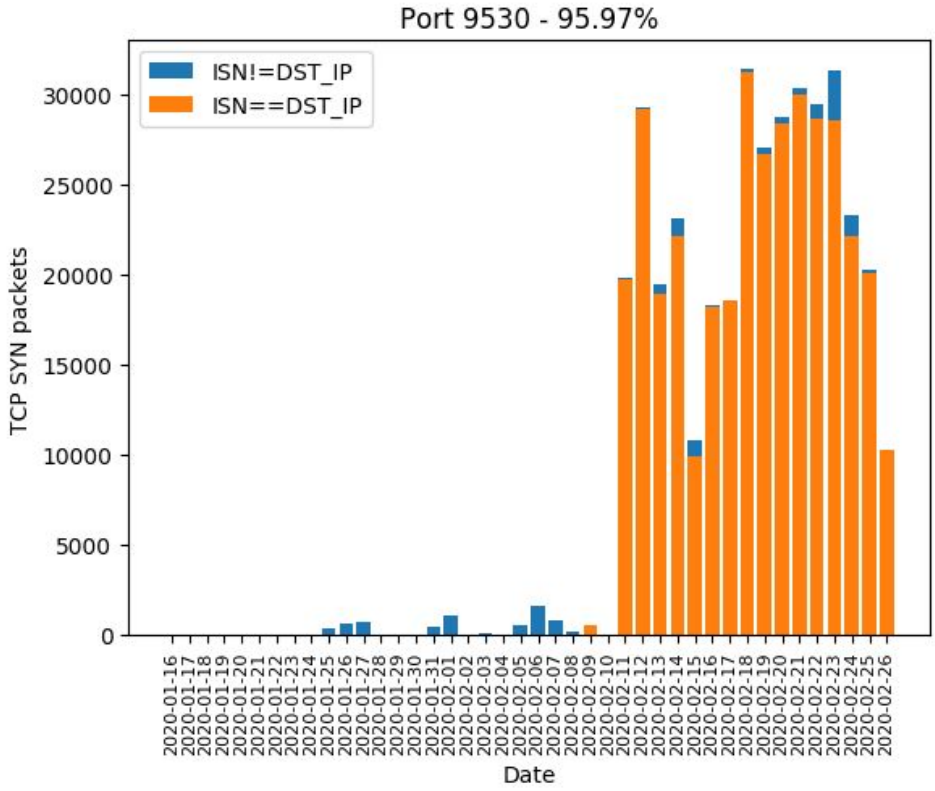
The reason why this port has been extensively targeted by botnet may be linked to the CVE-2017-17215[34]. This vulnerability affects the router Huawei HG532 with unpatched firmware, making it possible for a remote user to execute arbitrary shell commands. The official security notice is available on the Huawei website[35]. A detailed analysis can be found in a medium blog post[36].



[34] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-17215 visited on 2020-03-05
[35] https://www.huawei.com/en/psirt/security-notices/huawei-sn-20171130-01-hg532-en visited on 2020-03-05
[36] https://medium.com/@knownsec404team/huawei-hg532-series-router-remote-command-execution-analyzation-a531d96d5339 visited on 2020-03-05

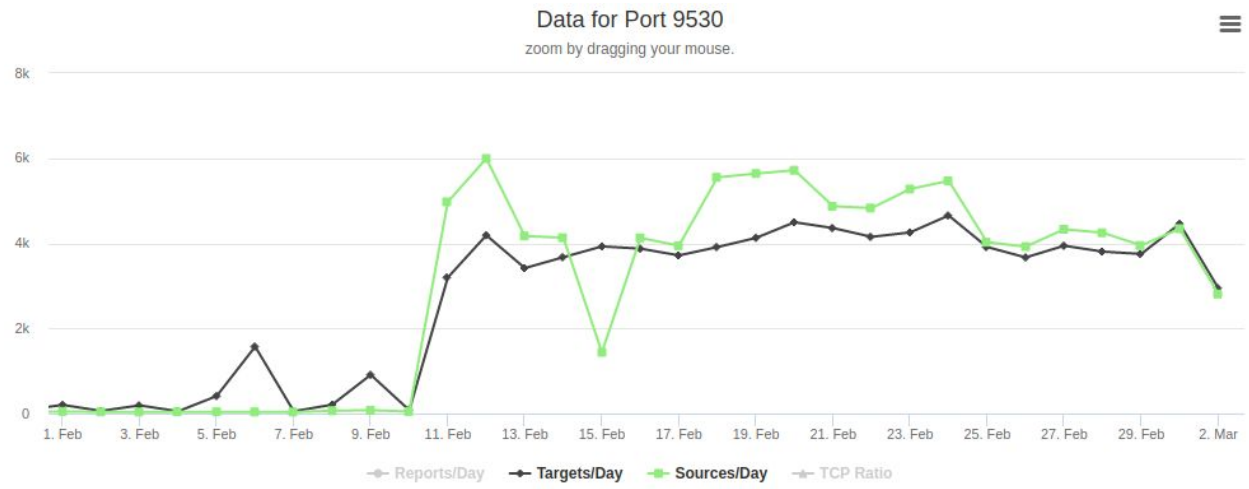
5.1.2 Port 9530

The period studied is very interesting for this port because the scan only started on 2020-02-11, as we can see on the graph. As an interesting coincidence, a few days earlier, on 2020-02-04, a 0-day vulnerability affecting Xiongmai DVR, NVR and security camera has been fully disclosed by Vladislav Yarmak on Habr website[37]. This vulnerability allows an attacker to open a Telnet daemon on port 9527. Connecting with default credentials, an attacker can execute shell commands as root. The official security notice is available on the Xiongmai website[38]. An interesting article from OSM Solutions website[39] explains why the vendor installed the backdoor in the first place: to allow people to reset their password. The backdoor has then been hidden better and better.



[37] https://habr.com/en/post/486856/ visited on 2020-03-05
[38] http://www.xiongmaitech.com/en/index.php/news/info/12/68 visited on 2020-03-05
[39] https://www.osm-s.com/en/2017/11/30/ip-camera-security-horror/ visited on 2020-03-05

As the data displayed on this graph is very atypical, I found it useful to compare it with another source of data to confirm that the scan is not targeting this particular infrastructure. We can see that it is not the case because the data from ICS SANS[40] shows the same pattern:



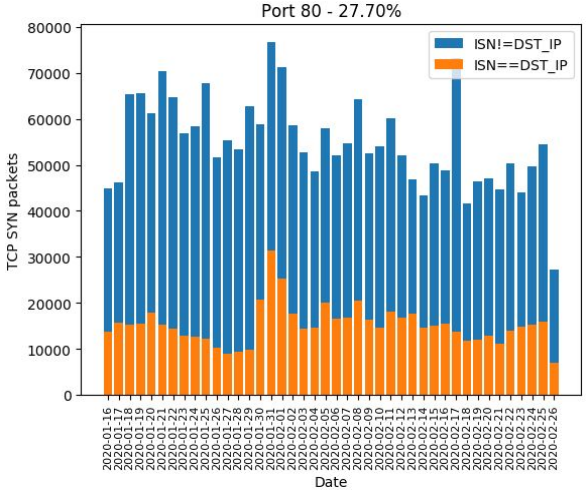
5.1.3 HTTP Ports 8[0-8]+

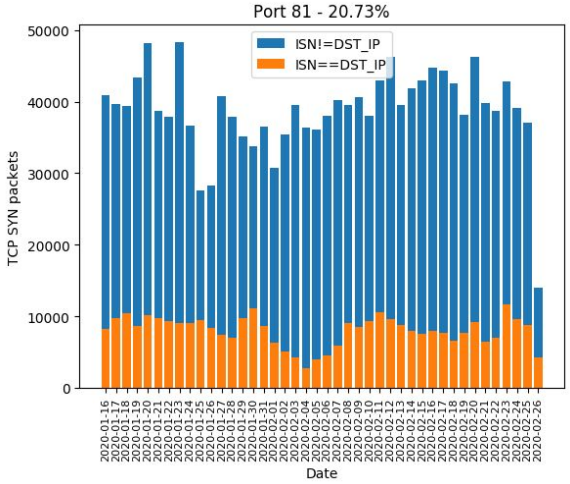
The HTTP port 80, along with its most common alternatives 81, 82, 83, 85, 88, 8000, 8080, 8081 are all (mainly) targeted for the same purpose: they serve an HTTP server. On managed devices, this HTTP server often gives access to administration web pages.

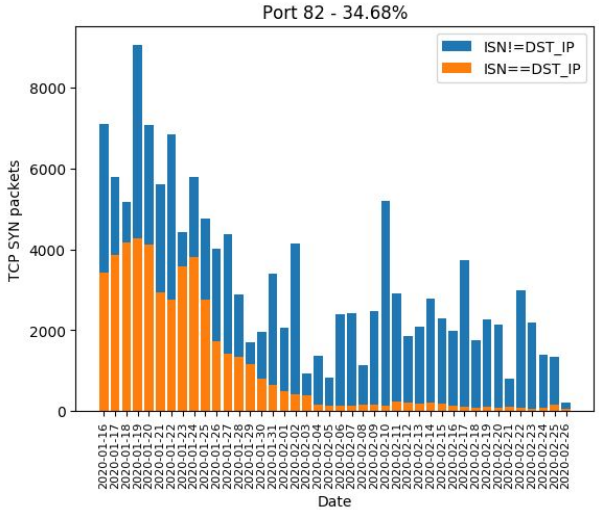
The first possible exploitation is to use the default password. According to a blog post from Positive Security[41], 15 devices out of 100 still have their default credentials. Whether this number is correct or not, Mirai' story showed us that a big number of devices can be exploited this way. The second possible exploitation is exploitation of the software itself. The HTTP server can be old and contain public vulnerabilities, as we will see for port 4567.

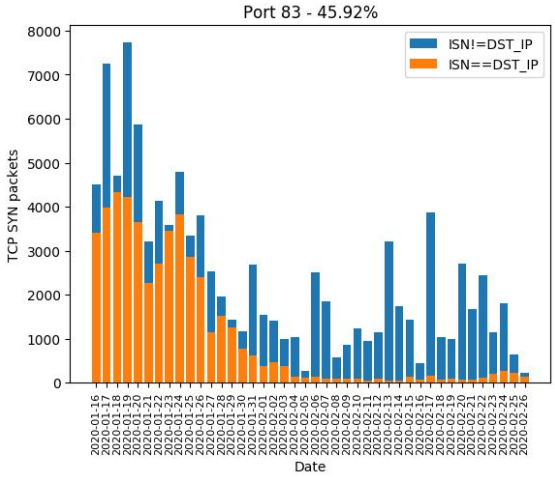
As we can see on the graphs, the interest for ports 82, 83, 85 and 8081 dropped in February. Due to the newly released vulnerabilities and exploits, we assume the botnet needs to regularly add new ports to scan. One way to keep infecting at a good rate is then to remove the less efficient ports which could be 82, 83, 85 and 8081.

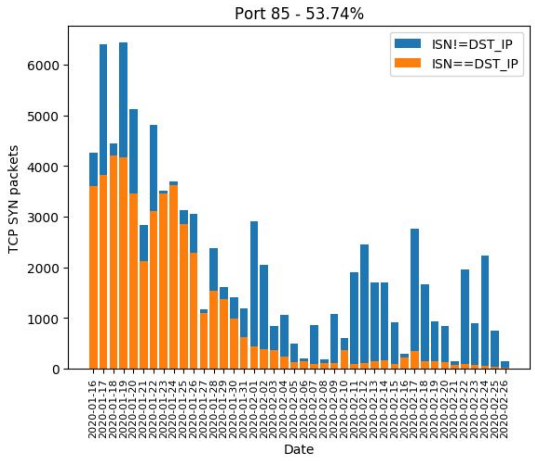
[40] https://isc.sans.edu/port.html?port=9530 visited on 2020-03-05.
[41] http://blog.ptsecurity.com/2017/06/practical-ways-to-misuse-router.html visited on 2020-03-05

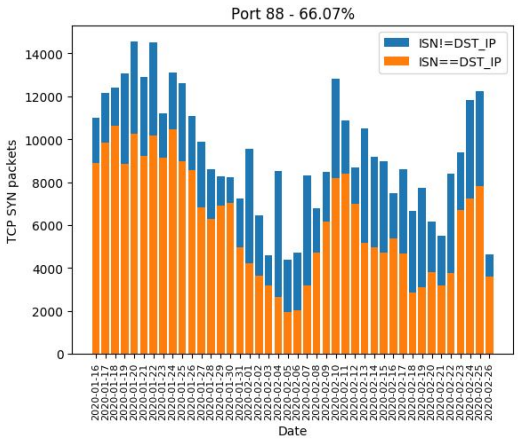

Port 80 - 27.70%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date

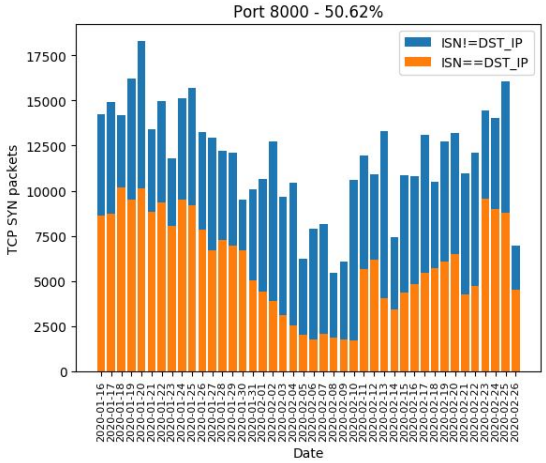

Port 81 - 20.73%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date

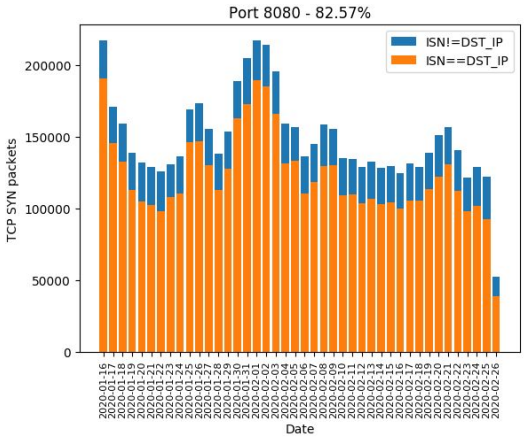

Port 82 - 34.68%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date

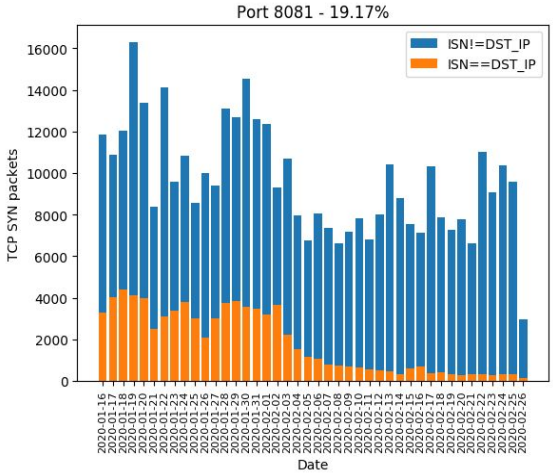

Port 83 - 45.92%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date


Port 85 - 53.74%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date


Port 88 - 66.07%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date


Port 8000 - 50.62%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date


Port 8080 - 82.57%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date

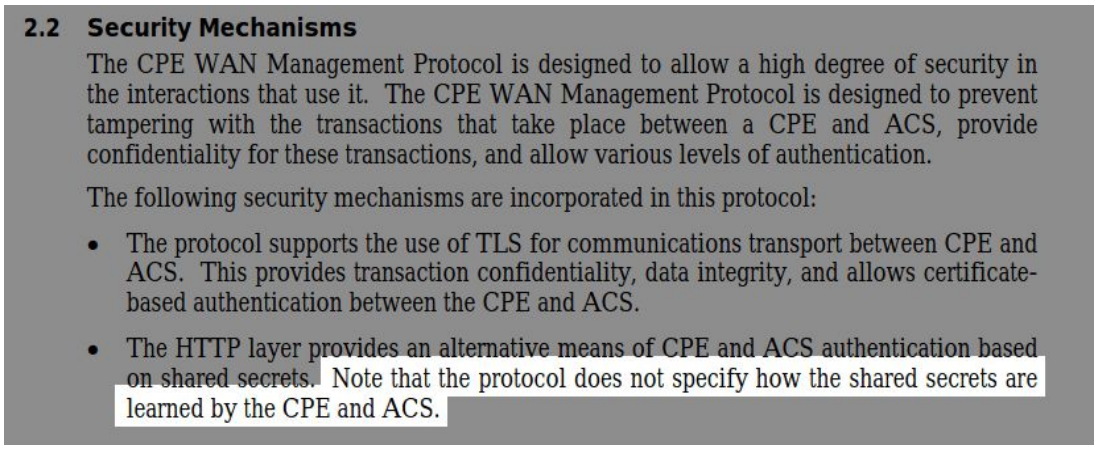

Port 8081 - 19.17%
ISN!=DST_IP
ISN==DST_IP
TCP SYN packets
Date

5.1.4 Port 4567

This port is used by the protocol Technical Report 069 or TR-069. This protocol is used for routers remote administration and firmware upgrades. The specification, updated in 2018, can be found on the Broadband Forum[42].

According to ISC[43], it is used by Verizon and other ISPs having Actiontec routers. We can find really old complaints about that back to 2007 for Verizon[44], 2010 for BT[45], 2014 for Century Link [46] or more recently in 2019 for Plusnet[47]. This feature looks quite standard and is widely used.

What about its security? Here's an overview from the specification:



2.2 Security Mechanisms

The CPE WAN Management Protocol is designed to allow a high degree of security in the interactions that use it. The CPE WAN Management Protocol is designed to prevent tampering with the transactions that take place between a CPE and ACS, provide confidentiality for these transactions, and allow various levels of authentication.

The following security mechanisms are incorporated in this protocol:

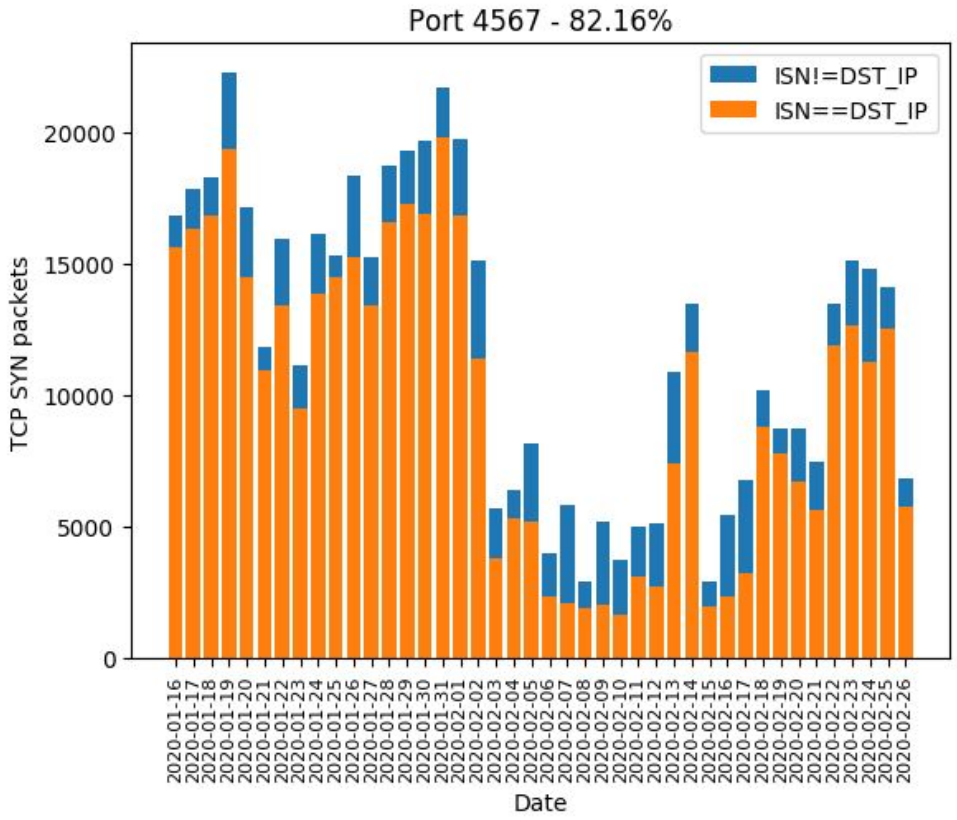
- The protocol supports the use of TLS for communications transport between CPE and ACS. This provides transaction confidentiality, data integrity, and allows certificate-based authentication between the CPE and ACS.
- The HTTP layer provides an alternative means of CPE and ACS authentication based on shared secrets. Note that the protocol does not specify how the shared secrets are learned by the CPE and ACS.

The protocol makes use of a shared secret between the Customer Premise Equipment (CPE) and the Auto-Configuration Server (ACS) but does not explain how to exchange the secret. As the secret sharing is not standard, not all ISPs will have the same implementation and some of them may take the easy way out to use a common default password for all their devices.

---

[42] https://www.broadband-forum.org/technical/download/TR-069_Amendment-6.pdf visited on 2020-03-05
[43] https://isc.sans.edu/port.html?port=4567 visited on 2020-03-05
[44] https://www.dslreports.com/forum/r19531564-Port-4567-The-Evil-Port visited on 2020-03-05
[45] https://community.bt.com/t5/Archive-Staging/port-4567-backdoor/td-p/59703 visited on 2020-03-05
[46] https://superuser.com/questions/832571/port-4567-open-on-router-to-tram-traffic-should-i-worry visited on 2020-03-05
[47] https://superuser.com/questions/832571/port-4567-open-on-router-to-tram-traffic-should-i-worry visited on 2020-03-05

Even when using unique passwords, some devices make use of outdated web servers allowing authentication bypass. An analysis from Shahar Tal and Lior Oppenheim at the RSA Conference 2015 called “The Internet of TR-069 Things: One Exploit to Rule Them All” revealed that more than 13 million devices were using RomPager 4.07 (released in 2002). The presentation can be accessed through the Internet Archive[48]. An exploit is now publicly available since 2016[49].



I notice here a lack of interest between 2020-02-03 and 2020-02-21. Maybe the attacker wanted to scan some other ports instead.

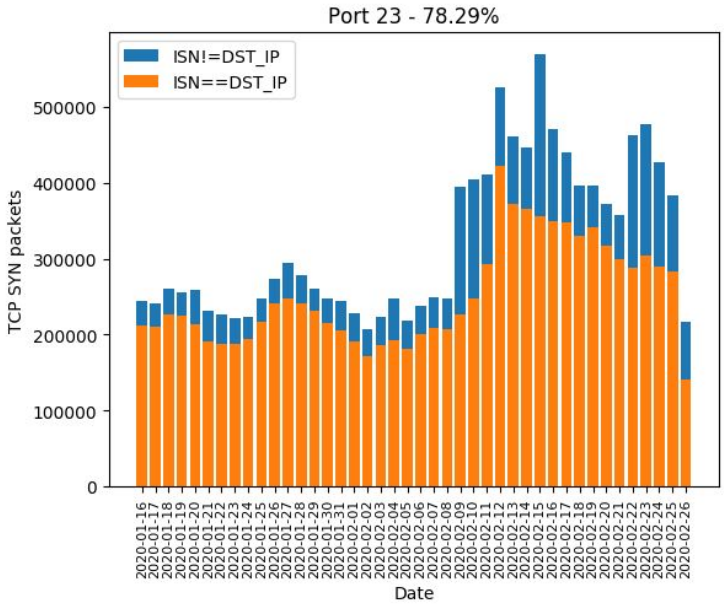
---

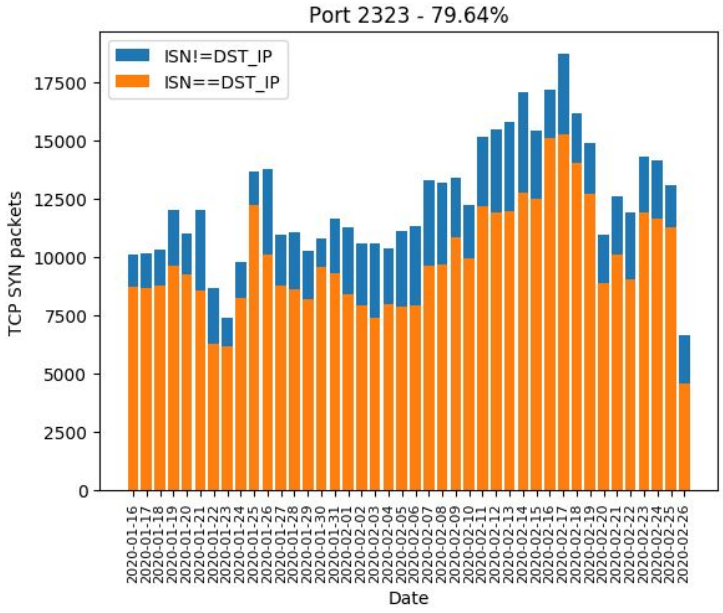
[48] https://web.archive.org/web/20190508230250/https://www.rsaconference.com/writable/presentations/file_upload/hta-r04-the-internet-of-tr-069-things-one-exploit-to-rule-them-all_final_copy1.pdf visited on 2020-03-05

[49] https://www.exploit-db.com/exploits/39739 visited on 2020-03-05

5.1.5 Telnet Port 23(23)

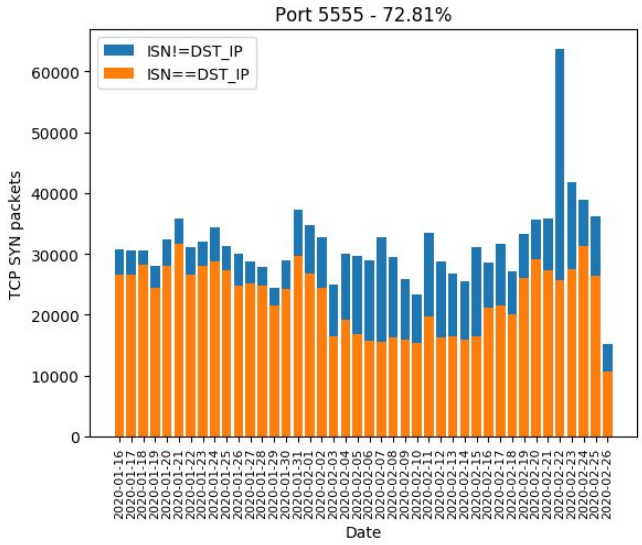
The telnet protocol was the first one to be targeted by Mirai. It is not surprising to see it is still very active today.





5.1.6 Port 5555

This port is used by the Android Debug Bridge (ADB)[50] daemon on Android devices. As its name suggests, it is used for debugging purposes. Stock Android does not allow debugging by default. The user has to enable USB debugging first in the hidden developer menu and then run **tcpip** command over USB to enable network debugging[51]. Moreover, Android 4.2.2, released in February 2013[52], added a security layer: the user has to unlock the device and accept the USB connection. It is hardly possible that a large number of users ran through all those steps to generate that interest:



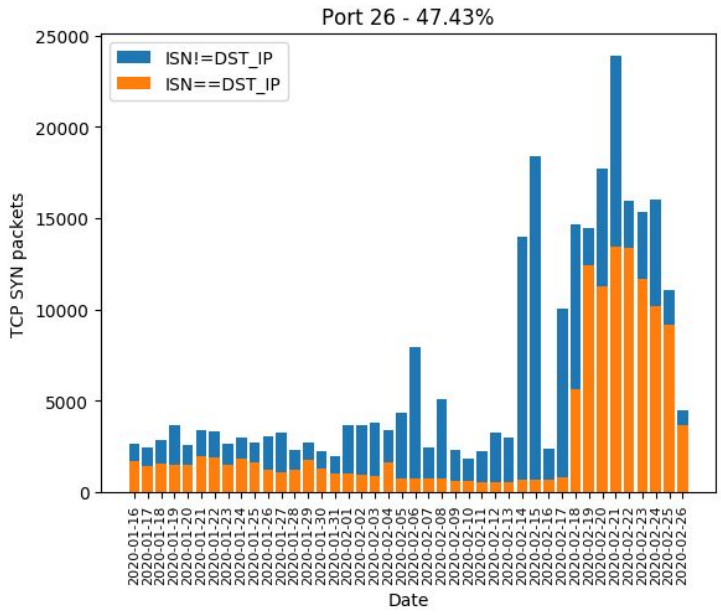
According to an article from Hui Wang on Netlab[53] and an analysis from Kevin Beaumont on DoublePulsar[54] in February 2018, some manufacturers shipped Android devices with the

[50] https://developer.android.com/studio/command-line/adb visited on 2020-03-05
[51] https://developer.android.com/studio/command-line/adb#wireless visited on 2020-03-05
[52] https://developer.android.com/studio/releases/platforms#revision-2-february-2013 visited on 2020-03-05
[53] https://blog.netlab.360.com/early-warning-adb-miner-a-mining-botnet-utilizing-android-adb-is-now-rapidly-spreading-en/ visited on 2020-03-05
[54] https://doublepulsar.com/root-bridge-how-thousands-of-internet-connected-android-devices-now-have-no-security-and-are-b46a68cb0f20 visited on 2020-03-05

network ADB bridge enabled and unauthenticated. Most devices are phones and TVs. Massive scans targeting this port started back in February 2018. At that time, the malware was using infected devices to mine cryptocurrencies like many others [Metongnon2018]. I doubt that a TV or a smartphone is very efficient for this job, but maybe this was profitable on a large scale. We can also find other analyses from TrendMicro[55] and from ISC[56] explaining that the botnet was still active in July 2018.

5.1.7 Port 26

Port 26 is also interesting during the studied period because we can see a huge increase in traffic from 2020-02-18. According to an analysis from the Internet Storm Center[57], the scanner is trying to connect to a Telnet-like terminal because of some network devices that expose a Telnet server on this port.



55 https://blog.trendmicro.com/trendlabs-security-intelligence/open-adb-ports-being-exploited-to-spread-possible-satori-variant-in-android-devices/ visited on 2020-03-05

56 https://isc.sans.edu/diary/Worm+%28Mirai%3F%29+Exploiting+Android+Debug+Bridge+%28Port+5555tcp%29/23856 visited on 2020-03-05

57 https://isc.sans.edu/forums/diary/Next+up+whats+up+with+TCP+port+26/25564/ visited on 2020-03-05

5.1.8 Other ports

I have analysed enough ports to show that Mirai-like scans now target a very important range of ports and security vulnerabilities. This shows the evolution of Mirai-like scans and of course, does not make an exhaustive list of all the newly targeted ports.

As released publicly in August 2016, Mirai was only targeting port 23 and 2323. This is not the case any more. This can be explained either by the fact that some Mirai variants evolved to include new exploitation methods, either by the fact that some other botnets have included the Mirai port scanning code. In both cases, the scanning code used by those new botnets should be very similar to the original Mirai one.

5.1.9 D4 Server module

Those statistics are not only interesting for this specific period. It would be much more interesting to get those graphics on a regular basis. I developed a D4 server module to achieve this goal. The code is available on GitHub[58]. When it is active, this module creates statistics from packet captures and saves them inside a local Redis database. One script creates the graphics from the statistics inside the database. An example of a cron job to run this script everyday is available in the README of the GitHub repository.

The next step for this module could be to send the aggregated data somewhere else. For example, people could send this kind of data to CERTs to make statistics at a larger scale. This module could also be extended to detect other scanning techniques.

5.1.10 Summary on common vulnerabilities

As we can see, a lot of devices have a default backdoor. From the vendor perspective, the reasons can be multiple. It can be there for security purposes - the vendor can this way update the device remotely. It can be there for usability purposes - the user can easily reset his password with an easy-to-use tool. This is fine as long as no one knows about it but once the information about this backdoor is found out and revealed on the internet, it becomes a serious

[58] https://github.com/D4-project/analyzer-d4-isn visited on 2020-02-28

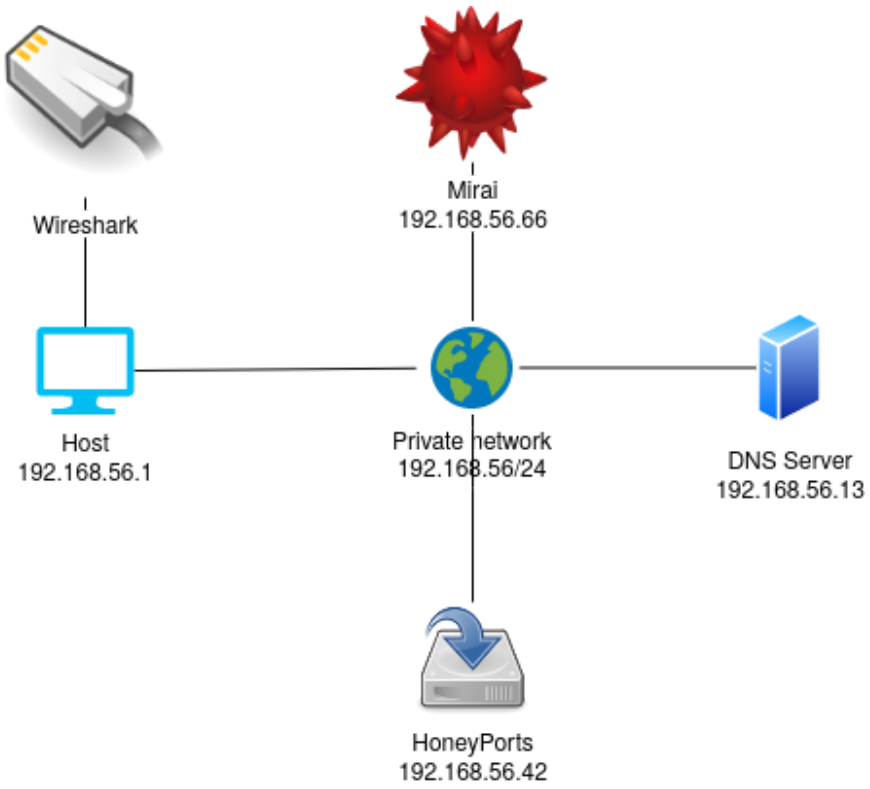
security issue for those devices. Some vendors try to hide the backdoor better using some customized port knocking or by adding a telnet captcha[59] as we can see on this screenshot:

Of course this pattern of security by obscurity is very discouraged[60].

[59] http://blog.ptsecurity.com/2017/06/practical-ways-to-misuse-router.html visited on 2020-03-05
[60] "Guide to General Server Security" (PDF). National Institute of Standards and Technology. July 2008. Retrieved 2020-03-05. “Open Design - System security should not depend on the secrecy of the implementation or its components.”

5.2 Setup

To run the tests, a virtual environment has been set up using the popular virtualization software VirtualBox 5.2.42. The host hardware is a Lenovo Thinkpad T490s with an Intel CPU i7-8665U and 32Go of RAM. The OS is Ubuntu 18.04.5 with kernel 5.4.0-42-generic. Three virtual machines or VMs have been allocated 50% of 1 CPU core and 1024Mo of RAM. VMs are running Ubuntu 18.04.5 with kernel 4.15.0-112-generic and are connected to a private network through a VirtualBox host-only adapter. Here is a schema of the network:



To access the internet, install or update packages, an additional interface is added to the VM. To avoid any trouble during the tests, VMs do not have internet access. The network is managed by VirtualBox, running a DHCP server distributing IP addresses. To have a consistent setup, I configured a static IP address and custom network parameters for each VM.

To be able to monitor host and VMs performance such as CPU, RAM, disks and network usage, I installed cockpit 215 on all of them. Cockpit is a pretty lightweight tool interfacing with systemd and displaying nice graphics in a web UI.

5.2.1 Host 192.168.56.1

The host also has access to the private network. This allows me to monitor the VMs using the cockpit HTTP server running on the host and accessing it via the host web browser. Firefox 79 was used to access cockpit web UI.

5.2.2 DNS server 192.168.56.13

A DNS server has been set up using BIND 9.11.3. Because there is no internet access, the DNS server does not know about IP addresses linked to domain names. To avoid the client from receiving a DNS error, the DNS server replies to all domain A requests with the fixed IP address 192.168.56.42. Here is the configuration file for BIND9:

```
 GNU nano 2.9.3                                          /etc/bind/db.single-ip

$TTL    60
@       IN      SOA     . hostmaster.localhost. (
                        20200403        ; Serial
                          604800        ; Refresh
                           86400        ; Retry
                         2419200        ; Expire
                          604800 )      ; Negative Cache TTL
;
@       IN      NS      ns
ns      IN      A       192.168.56.13
*       IN      A       192.168.56.42
```

5.2.3 HoneyPorts 192.168.56.42

The name comes from the concatenation of honeypot and ports. This VM acts as a honeypot for TCP traffic. A single server can be used to listen to all TCP ports and accept all clients connections. This is useful because the client does not receive an error and sends the data, allowing traffic monitoring. A ncat server is started on port 4242 to accept all the clients:

```
ncat -l 4242 -k
```

Those iptables rules are used to redirect all non-used ports to the port 4242:

```
sudo iptables -A OUTPUT -p tcp -m tcp --tcp-flags RST RST -j DROP # block TCP_RST

# persistent?
sudo ipset create used_ports bitmap:port range 0-65535
sudo ipset add used_ports 22
sudo ipset add used_ports 9090

# route to specific port
sudo iptables -t nat -A PREROUTING -p tcp -m set ! --match-set used_ports dst -j REDIRECT --to-ports 4242
```

5.2.4 Mirai 192.168.56.66

This VM contains the Mirai botnet. The network is monitored thanks to the VirtualBox “nictrace” feature. This method saves all the traffic from this VM. The Mirai botnet has been compiled thanks to the instructions from CDXY[61]. The malware is not persistent when the infected device is rebooted [Sinanovic2017]. The VM is rebooted before each test to ensure only one instance of the malware is running. No restoration from a saved snapshot is needed in this case.

[61] https://www.cdxy.me/?p=746 visited on 2020-04-15

5.3 Mirai in practice

5.3.1 No response

As we have seen in section 4.2.3, the more Mirai spend time making connections and bruteforcing credentials, the less SYN probes are sent. In the first test, we consider the scenario where Mirai does not receive any response. No response means that Mirai will not spend much time checking SYN+ACK or brute forcing credentials and should be able to achieve its maximum throughput. During this first test, Mirai delivered **157** TCP SYN packets per second. This number is relatively close to the maximum defined in the **scanner.h** file. The SCANNER_MAX_CONNS indicates the maximum number of telnet servers to bruteforce at the same time.

```
8  #define SCANNER_MAX_CONNS    128
9  #define SCANNER_RAW_PPS      160
```

We have also seen that Mirai targets less than 3 422 552 064 IPv4 addresses. At the measured throughput, all those addresses can be scanned in 1 day with 253 devices. During an analysis from December 2016[62], the number of Mirai infected devices was around 600 000. That means a random IPv4 address was receiving, on average, 1 scan every 36 seconds. We also know that the brute forcing software checks 10 sets of credentials at each scan, among 62 available sets. After 5 minutes, a device had received on average 83 login attempts and, in the case it was vulnerable, there was a high chance for it to be effectively infected. The same time slot applies if the owner resets its device to the default configuration while the device is online. When we look at those numbers, we can understand why the Mirai botnet was spreading so fast.

5.3.2 Accept all connections

In the second test, I tried to interact a bit with the Mirai botnet. First of all, I started a netcat server on port 4242 using the HoneyPorts method, which means this server also listens on port 23 and 2323. I used a python script to send SYN+ACK packets with the right acked sequence

[62] https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/ visited on 2020-03-25

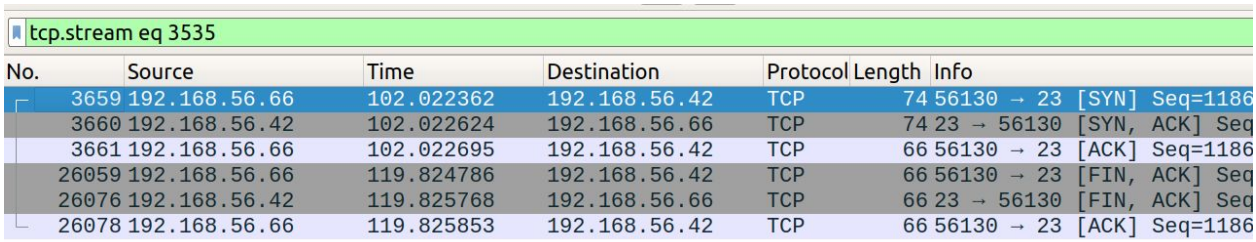
number to Mirai. The malware interpreted those as responses to SYN stateless scan and opened 1 connection back for each received packet. Here is the script:

```
#!/usr/bin/env python
import time
import sys
from scapy.all import *

def ip_to_int(ip):
  parts = ip.split('.')
  i = 0
  for p in parts:
    i *= 256
    i += int(p)
  return i

if len(sys.argv) < 2:
  exit('missing destination port')
port = int(sys.argv[1])
if port < 1 or port > 65535:
  exit('invalid port')

src = '192.168.56.42'
host = '192.168.56.66'
pps = 100

seq = ip_to_int(src)+1
p = IP(src=src,dst=host)/TCP(sport=23,dport=port,ack=seq,flags='SA')
send(p, inter=1/pps, count=10*pps)
```

The parameter of the script is the port from which Mirai sends its TCP packets. It is selected randomly when the program starts. On a production system, you can see it when you receive a SYN packet from the scanner. In this case, I found the port using the network capture.

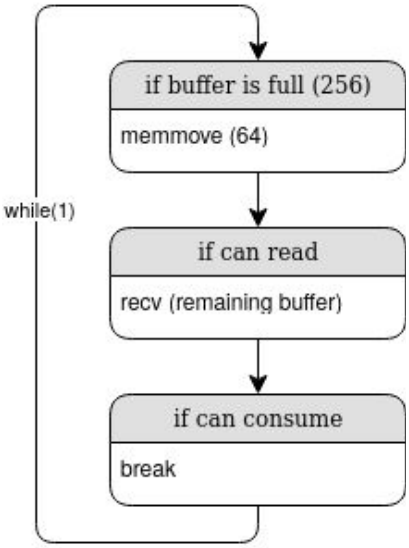
I do not expect any impact on the scanner in this test because not so many instructions are executed to initiate a connection. As expected, I did observe a similar rate of SYN packets.

As visible on one a screenshot of the packet capture, the connections is kept alive:



Mirai tries 10 different sets of credentials unless one succeeds. During credentials brute forcing, if the connection is stuck and the last byte was received more than 5 seconds ago, the connection is closed. Then it is opened again and again 10 times. Thus, the minimum time that an unauthenticated target stays in the reserved 128 slots is 50 seconds. If I just send SYN+ACK packets to Mirai and it is able to connect back, this will occupy one slot among the 128 available ones for 50 seconds and Mirai will not bruteforce other devices.

5.3.3 Null bytes

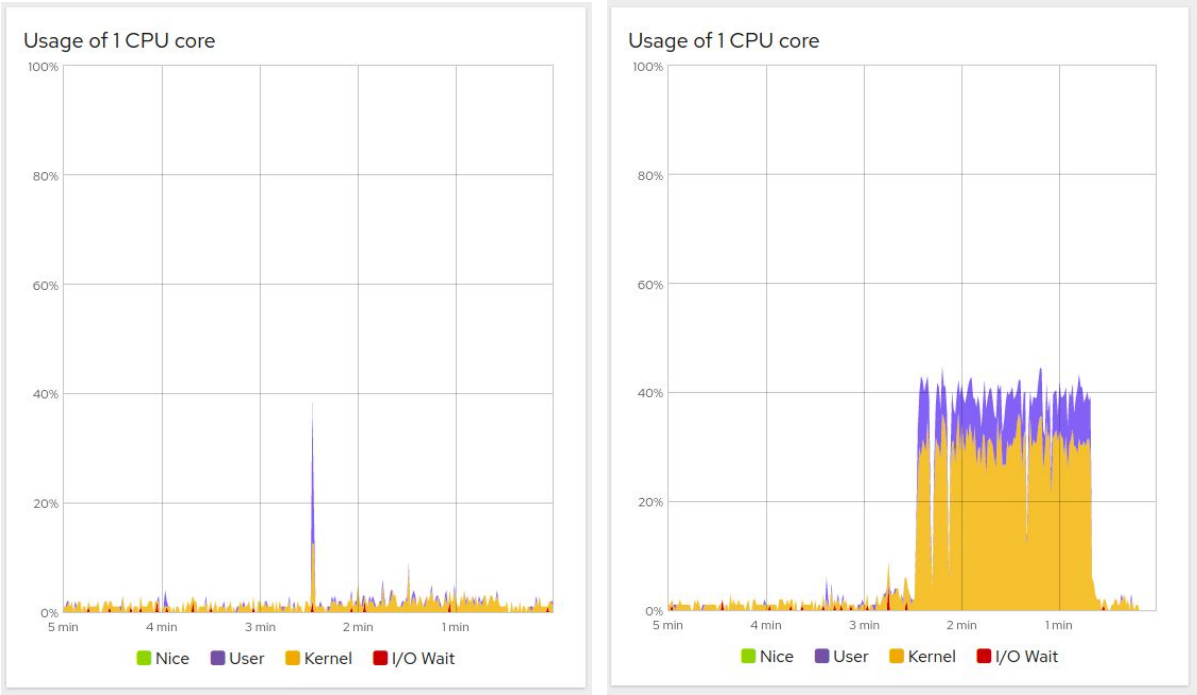


Looking at the brute forcing code, I found out that after the establishment of the connection, Mirai was reading, from the network, 256 bytes at first in a fixed size buffer and then 64 bytes at a time. The reason for this is to be able to read more bytes using a fixed size buffer, and without consuming more memory. After the first 256 bytes, if Mirai cannot "consume" the bytes then the first 64 bytes of the buffer are dropped and the next 64 bytes are read from the network. This behavior lasts until Mirai can consume the buffer or no more data can be read from the network.

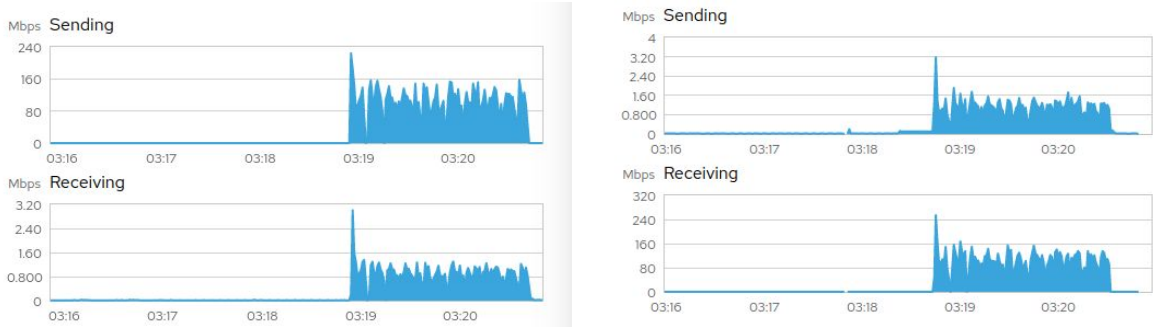
I am abusing this behavior by sending a big enough amount of data. Mirai is making a lot of calls to the memmove and recv functions, spending a lot of time in context switches between user and kernel mode. In this test, we send 1024*1024 null bytes (1Mo) once Mirai opens the connection on port 23. Here is the python code to send the data:

```
#!/usr/bin/env python
import socket
import _thread as thread

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

s.bind(('192.168.56.42', 4242))
s.listen(1000)


def new_client(conn, addr):
    conn.send(b'0x00'*1024*1024)
    conn.close()

try:
  while True:
    conn, addr = s.accept()
    print('Connection from '+str(addr))
    thread.start_new_thread(new_client, (conn, addr))
except KeyboardInterrupt:
  s.close()
```

I have run this test 3 times and got quite different results: only 64, 108 and 96 SYN packets per second were sent by Mirai. Now let's have a look at the impact of this test on the VMs:



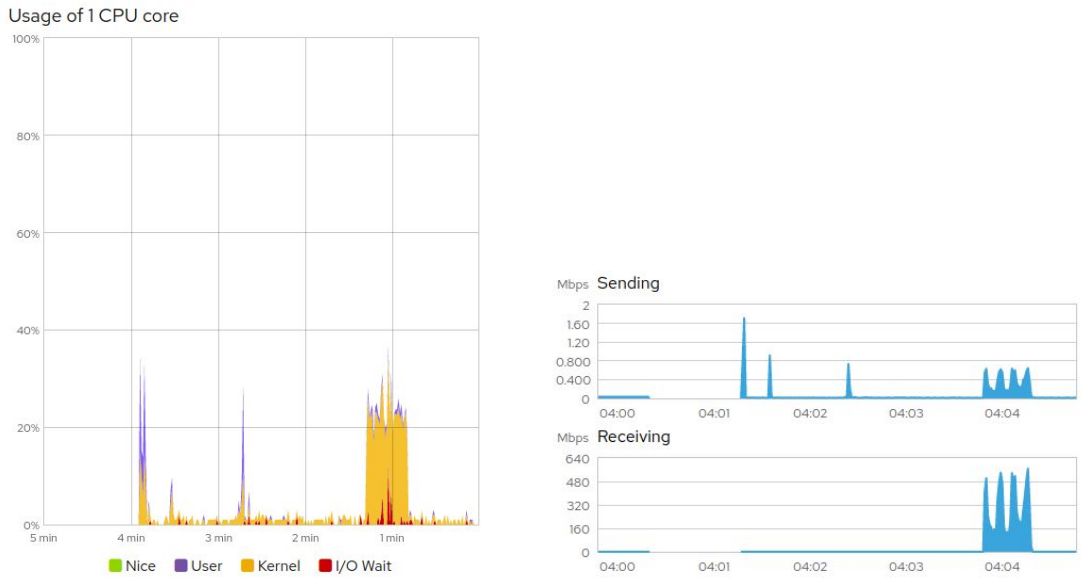
CPU usage for respectively the offensive VM and the Mirai VM



Network usage for respectively the offensive VM and the Mirai VM

As we can see on the network graphics, the bandwidth used is quite important here. Around 120Mbps is going from the offensive VM to the Mirai VM. The Mirai VM's CPU is as expected,

under heavy load. One could argue that maybe the network is causing the CPU usage so I also made a test sending a big 1Go file to the Mirai VM to know about the impact of the network on the CPU. Here is the result:



CPU and network usage of the Mirai VM during network file transfer

As we can see, the network usage is on average 350Mbps instead of 120 during the previous test and the CPU usage is only around 25% instead of 40% during the previous test. With those results, we can see that the CPU usage of the previous test is not only due to the network traffic.

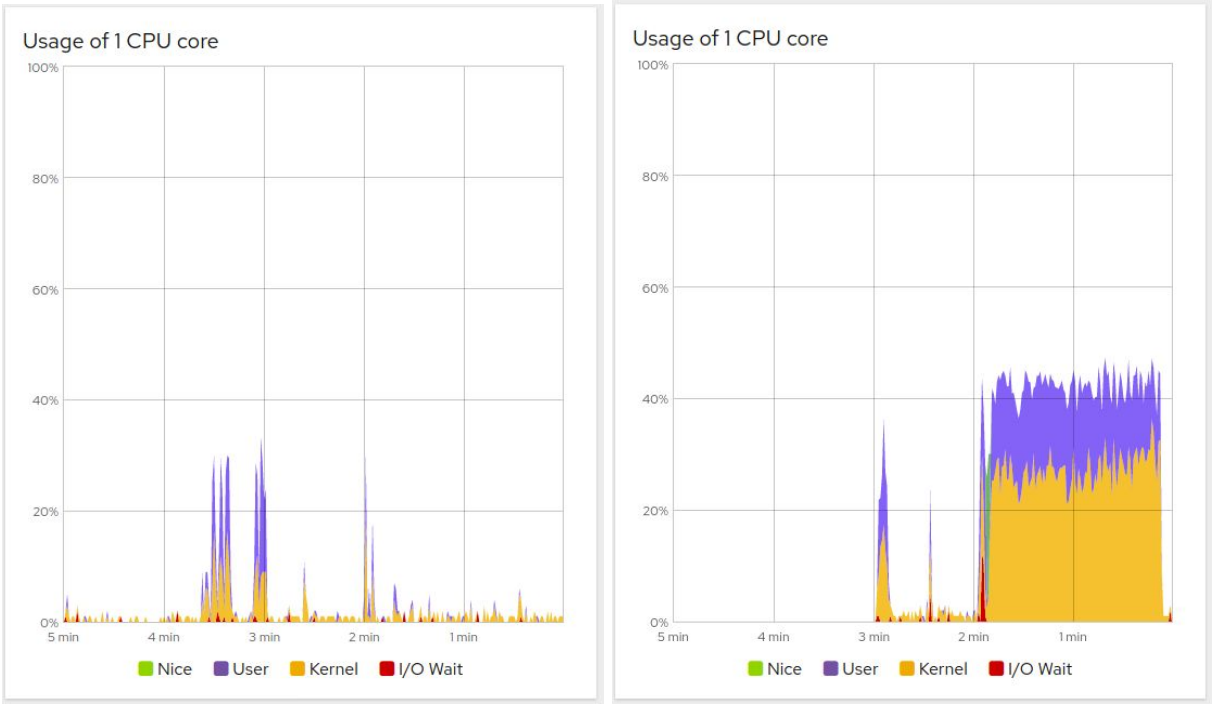
Even if the bandwidth usage in this test, 120Mbps, is too high to make it possible over the internet, it is an interesting result because it means only one person responding to Mirai scans can reduce the impact of this one. Our theory is verified in practice. In this case, the impact of the scanner was decreased by about a half.

5.3.4 0xFF bytes

Mirai maintains a state for each connection it is trying to bruteforce. The states can be “waiting username” or “waiting password”. To change from one state to the next one, heuristics are used on the received bytes. Each state has a particular heuristic to go to the next state. For example,

to enter the state “waiting password” from the state “waiting username”, Mirai is looking for one of the the following characters: “**:>$#**”. Those are indeed common characters to show the user he has to enter something in the terminal. Another part of this heuristic is looking for “assword”. We can understand that this matches both “Password” and “password”.

Those heuristics are more or less CPU intensive. I have run tests for all of them and found the more CPU intensive one is the “waiting username” heuristic. For the code to reach this heuristic, the easiest way is to send only 0xFF bytes. To be able to compare the results with the previous test, we sent the same amount of data (1Mo) on each connection opened by Mirai. Here are the performance graphics:



CPU usage for respectively the offensive VM and the Mirai VM

Network usage for respectively the offensive VM and the Mirai VM

During this test, 25 to 35 SYN packets were sent per second by the Mirai scanner. This is 5 times less than the original rate. As we can see, the network usage is reduced to around 60Mbps. This test is about 4 times better than the previous one because it requires about half the bandwidth and has about twice better results.

One difference we can notice on the CPU graphic is that this test uses a bit more user-mode code than the previous one.

6 Discussion

To run the tests, I used a high-end CPU which is fast and has a lot of optimizations. On the contrary, an IoT device is usually cheap and is likely to have a low-end CPU which would be able to execute much less instruction per second and would have fewer optimizations. This could give better results with less bandwidth.

6.1 Future work

This work mainly focuses on the Mirai bot scanner but there are other parts of the ecosystem that could be studied. For example, Mirai attack code could be used by people under attack. We could send fake Mirai client requests to the command & control server to overload it. We could send invalid IP+port+credentials combo to the report server.

The actual results are nice but they are still “theoretical” because they have been extracted from a virtual environment. The actual constraints on real devices could be different. The next step would be to try those tests on real devices such as old routers and security cameras.

The architecture used here is only x84_64. The behavior of the compiled binaries can be very different from one architecture to another. The context switches could have different costs on different architectures. To make an exhaustive documentation about the behavior of Mirai running on each supported architecture, those tests need to be done on every one of them.

All methods could be run in real life in a production environment. Though, sending data to those devices can generate a lot of load on the CPU, potentially making them less efficient for their usual tasks. Even though sending data to a device without saving anything received is privacy compliant [Zeitlin2015], I would suggest to address the legal aspect before doing it.

The studied code was released in 2016. Some attackers used the code as-is but some others may have improved it. Further research could be made on the actual sample collected today.

7 Conclusion

I have successfully reached the goals foreseen in the introduction. This work even opens new perspectives for further investigation such as the detection of port scanners, the exploitation of the Mirai botnet infrastructure, ...

I have made a comparison between the main port scanners used currently. Any actual implementation can be detected more or less easily. I created a D4 server module to make statistics about the Mirai scanner. My contribution allows the team of the D4 project at CIRCL to further improve detection mechanisms of network scanning.

I have studied the current status of the Mirai-like botnets and they are still very active today. I have made an analysis of Mirai's source code. I found out several weaknesses and I exploited them to demonstrate they can reduce the efficiency of the Mirai scanner. Indeed, it is possible to make it less active at brute forcing other devices and at scanning the internet for new targets.

Something that surprised me when looking at the state of the art is that the number of scientific papers are quite low compared to blog posts. Blog posts are faster to write and read but sadly, their longevity is shorter. When I realize a blog post about a research is not available any more today, it is a bit frustrating to me and it is a great loss to the scientific community. Thanks to the Internet Archive[63] (also called the wayback machine), I was able to access most of them. Unfortunately, not all web pages are saved so this technique was not always available. This project was a great help and I invite anyone reading this to support it and so, to save more pages in the future.

The subject is so large and interesting that it was not easy to stay focused on a specific point. I would like to thank all the people that supported me during the several months of research, including, but not limited to, my family, my friends, all the people at the CIRCL and more specifically Alexandre Dulaunoy and Jean-Louis Huynen, my supervisor from UCLouvain Ramin Sadre and the readers.

[63] https://archive.org/web/ visited on 2020-08-06

8 Bibliography

[Antonakakis2017] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, & Yi Zhou (2017). Understanding the Mirai Botnet. In 26th USENIX Security Symposium (USENIX Security 17) (pp. 1093–1110). USENIX Association.

[Lathrop2014] J. Lathrop and J. B. O'Kane, "A Case Study in Opportunity Reduction: Mitigating the Dirt Jumper Drive-Smart Attack," 2014 IEEE Joint Intelligence and Security Informatics Conference, The Hague, 2014, pp. 224-227, doi: 10.1109/JISIC.2014.41.

[Lo2018] Fang-Yi Lo, Nayara Campos (2018). Blending Internet-of-Things (IoT) solutions into relationship marketing strategies, doi: 10.1016/j.techfore.2018.09.029.

[Metongnon2018] Metongnon, L., & Sadre, R. (2018). Beyond Telnet: Prevalence of IoT Protocols in Telescope and Honeypot Measurements. In Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity (pp. 21?26). Association for Computing Machinery.

[Mokaddem2017] Mokaddem, Sami. Measurements of compromised IoT devices from blackhole and honeypot. Ecole polytechnique de Louvain, Université catholique de Louvain, 2017. Prom. : Sadre, Ramin. http://hdl.handle.net/2078.1/thesis:10671

[Shannon2004] C. Shannon and D. Moore, "The spread of the Witty worm," in IEEE Security & Privacy, vol. 2, no. 4, pp. 46-50, July-Aug. 2004, doi: 10.1109/MSP.2004.59.

[Sinanovic2017] H. Sinanovic and S. Mrdovic, "Analysis of Mirai malicious software," 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, 2017, pp. 1-5, doi: 10.23919/SOFTCOM.2017.8115504.

[Stone-Gross2010] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel and G. Vigna, "Analysis of a Botnet Takeover," in IEEE Security & Privacy, vol. 9, no. 1, pp. 64-72, Jan.-Feb. 2011, doi: 10.1109/MSP.2010.144.

[Wagner2017] Wagner, Cynthia & Dulaunoy, Alexandre & Wagener, Gérard & Mokaddem, Sami. (2017). An extended analysis of an IoT malware from a blackhole network.

[Zeitlin2015] Sam Zeitlin (2015). Botnet Takedown and the Fourth Amendment.

[Zhang2014] Z. Zhang, M. C. Y. Cho, C. Wang, C. Hsu, C. Chen and S. Shieh, "IoT Security: Ongoing Challenges and Research Opportunities," 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, Matsue, 2014, pp. 230-234, doi: 10.1109/SOCA.2014.58.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**