

DuneQuixote campaign targets Middle Eastern entities with “CR4T” malware



Authors

- Expert **GReAT**

Introduction

In February 2024, we discovered a new malware campaign targeting government entities in the Middle East. We dubbed it “DuneQuixote”; and our investigation uncovered over 30 DuneQuixote dropper samples actively employed in the campaign. These droppers, which exist in two versions – regular droppers and tampered installer files for a legitimate tool named “Total Commander”, carried malicious code to download an additional payload in the form of a backdoor we call “CR4T”. While we identified only two CR4T implants at the time of discovery, we strongly suspect the existence of others, which may be completely different malware.

The group behind the campaign took steps to prevent collection and analysis of its implants and implemented practical and well-designed evasion methods both in network communications and in the malware code.

Initial dropper

The initial dropper is a Windows x64 executable file, although there are also DLL versions of the malware sharing the same functionality. The malware is developed in C/C++ without utilizing the Standard Template Library (STL), and certain segments are coded in pure Assembler. All samples contain digital signatures, which are, however, invalid.

Upon execution, the malware initiates a series of decoy API calls that serve no practical purpose. These calls primarily involve string comparison functions, executed without any conditional jumps based on the comparison results.

```
mov    rcx, rax          ; lpSystemTime
call   cs:GetSystemTime

lea    rdx, aCuandoElBuenoD ; " Cuando el bueno de armstrong o
lea    rcx, aRetornar    ; "Retornar"
call   cs:lstrcmpA

lea    rdx, aSiempreTeEncue ; " Siempre te encuentro, oigo tu voz
lea    rcx, aSiempreHabr ; " Siempre habr"
call   cs:lstrcmpA

lea    rdx, aAMiPadrePorque ; "A mi padre,
lea    rcx, aDeTantoAmaryAn ; "DE tanto amar y andar salen los li
```

Useless function calls

The strings specified in these functions are snippets from Spanish poems. These vary from one sample to another, thereby altering the signature of each sample to evade detection using traditional detection methodologies. Following the execution of decoy functions, the malware proceeds to construct a structure for the necessary API calls. This structure is populated with offsets of Windows API functions, resolved utilizing several techniques.

Initially, the malware decrypts the names of essential Windows core DLLs using a straightforward XOR decryption algorithm. It employs multiple decryption functions to decode strings, where a single function might decrypt several strings. However, in our analysis, we observed samples where each string was decrypted using a dedicated function, each employing a slightly varied decryption algorithm.

```
__fastcall DecrXor1(_BYTE *encrypted_bytes)
{
    unsigned int counter; // r8d
    _BYTE *encrypted_bytes_1; // rdx

    counter = 0;
    encrypted_bytes_1 = encrypted_bytes;
    do
    {
        ++counter;
        *encrypted_bytes_1 ^= encrypted_bytes_1[13];
        ++encrypted_bytes_1;
    }
    while ( counter < 0xD );
    return encrypted_bytes;
}
```

String decryption algorithm

Once the necessary strings have been decrypted, the malware uses a standard technique for dynamically resolving API calls to obtain their memory offsets by:

- retrieving the offset of the Process Environment Block (PEB);
- locating the export table offset of *kernel32.dll*;
- identifying the offset for the GetProcAddress function.

In the process of obtaining the PEB offset, the malware first decrypts the constant *0x60*, which is used to locate the PEB64 structure. This approach is of particular interest because, typically, malicious samples or shellcode utilizing this technique opt for a hardcoded plain text constant value for this purpose.

```
call    Decrypt_offset
movsx   ecx, byte ptr [rbp+arg_8]
mov     rax, qs:[rcx]
```

Getting PEB structure offset

Next, the malware begins to populate the previously created structure with the offsets of all required functions.

The dropper then proceeds to decrypt the C2 (Command and Control) address, employing a unique technique designed to prevent the exposure of the C2 to automated malware analysis systems. This method involves first retrieving the filename under which the dropper was executed, then concatenating this filename with one of the hardcoded strings from Spanish poems. Following this, the dropper calculates the MD5 hash of the concatenated string, which is then used as a key for decrypting the C2 string.

```
result OK = calculate_md5(v39, (struct_some_heap_5_s71 *)filename_and_spanish_str, data_len, (__int64)&MD5_key);
if ( (!DWORD)result OK )
{
    buff_decrypted_c2 = AllocatesHeap(58164);
    *p_decrypted_c2 = buff_decrypted_c2;
    for ( lenght = 0i64; lenght < 0x39; ++lenght )
        *( _BYTE *) (buff_decrypted_c2 + lenght) = *((_BYTE *) &MD5_key + (lenght & 0xF)) | *((_BYTE *) (lenght + encrypted_c2)) & ~*((_BYTE *)
        *( _BYTE *) (buff_decrypted_c2 + 0x39) = 0;
    return 1i64;
}
return result OK;
```

C2 decryption algorithm

Following the decryption of the C2 string, the malware attempts to establish a connection with the C2 server using a specifically hardcoded ID as the user agent to download the payload. During our research of the C2 infrastructure, we found that the payload remains inaccessible for download unless the correct user agent is provided. Furthermore, it

appears that the payload may only be downloaded once per victim or is only available for a brief period following the release of a malware sample into the wild, as we were unable to obtain most of the payload implants from active C2 servers.

Once the payload is downloaded into the process's memory, the dropper performs a verification check for the "M" (0x4D in hexadecimal) magic byte at the start of the memory blob. This check likely serves to confirm that the payload has an MZ file signature, thereby indicating it is a valid executable format.

Total Commander installer dropper

The Total Commander installer dropper is created to mimic a legitimate Total Commander software installer. It is, in fact, the legitimate installer file, but with an added malicious file section (.textbss) and a modified entry point. This tampering results in invalidating the official digital signature of the Total Commander installer.

The installer dropper retains the core functionality of the initial dropper but with several key differences. Unlike the original dropper, it omits the use of Spanish poem strings and the execution of decoy functions. It also implements a series of anti-analysis measures and checks that prevent a connection to C2 resources, if any of the following conditions are true:

- a debugger is present in the system;
- known research or monitoring tools are among running processes;
- *explorer.exe* process has more than two instances
- any of the following processes are running:
 - "python.exe"
 - "taskmgr.exe"
 - "procmon.exe"
 - "resmon.exe"
 - "eventvwr.exe"
 - "process_hacker.exe"
- less than 8 GB RAM available;
- the position of the cursor does not change over a certain timeframe;
- disk capacity is less than 40 GB.

If any of the anti-analysis checks fail, the malware returns a value of 1. This specific return value plays a role in the decryption of the C2 server address. It triggers the removal of the first "h" from the beginning of the C2 URL ("https"), effectively changing it to "tpps". As a result, the altered URL prevents the establishment of a connection to the C2 server.

Memory-only CR4T implant

The "CR4T" implant is designed with the primary goal of granting attackers access to a console for command line execution on the victim's machine. Additionally, it facilitates the download, upload, and modification of files. The malware carries a PDB string in its code:

```
1 "C:\Users\user\Desktop\code\CR4T\x64\Release\CR4T.pdb"
```

That's why we dubbed it "CR4T".

Upon execution by the dropper, the implant initiates a *cmd.exe* process in a hidden window and establishes two named pipes to enable inter-process communication. It then configures the user agent for communication with the C2 server, embedding the hardcoded value "TroubleShooter" as the user agent name for requests to the C2.

```
lpThreadParameter_1 = lpThreadParameter;  
WinHttpOpen(L"TroubleShooter", 0, 0i64, 0i64, 0);  
ProcessHeap = GetProcessHeap();  
pbBinary_1 = (BYTE *)HeapAlloc(ProcessHeap, 0, 0x200ui64);  
pbBinary = pbBinary_1;
```

User-agent string

After that, the implant retrieves the computer name of the infected host as well as the username of the current user. Then it establishes a connection to the C2 server. This session provides interactive access to the command line interface of the victim's machine via the earlier mentioned named pipes. Commands and their outputs are encoded using Base64 before being sent and decoded after receiving.

After establishing the connection, the implant remains idle, awaiting an initial command from the C2 operator to activate the required functionality. This command is represented by a one-byte value, each one mapped to a specific

action on the infected system. These single character commands would likely make more sense for an English-speaking developer/operator than a Spanish-speaking one. i.e. "D" == Download, "U" == Upload (where a Spanish speaker might use "Cargar").

Command	Functionality
'C'(0x43)	Provide access to the command line interface via a named pipe.
'D'(0x44)	Download file from the C2
'U'(0x55)	Upload file to the C2
'S'(0x53)	Sleep
'R'(0x52)	Exit process
'T'(0x57)	Write to a file (T here possibly stands for a file-write <i>task</i>)

During our investigation, we discovered evidence of a PowerShell file that had been created using the "T" command:

```
1 "powershell -c \"Get-ScheduledTask | Where-Object {$_.TaskName -like 'User_Feed_Sync*' -and $_.State -eq 'Running'} | Select-Object TaskName\"
```

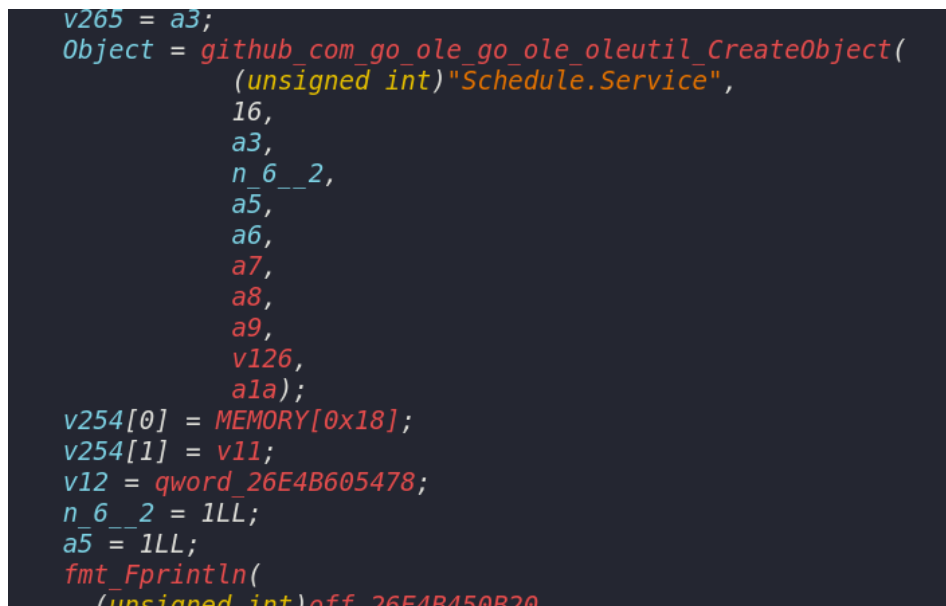
The threat actor was observed attempting to retrieve the names of all scheduled tasks on the infected machine beginning with "User_Feed_Sync". These scheduled tasks were probably created by the Golang version of CR4T for persistence purposes.

Memory-only Golang CR4T implant

We also discovered a Golang version of the CR4T implant, which shares similar capabilities with the C version and has a similar string related to the internal naming:

```
1 "C:/Users/user/Desktop/code/Cr4tInst/main.go"
```

This variant provides a command line console for interaction with infected machines, as well as file download and upload capabilities. It also possesses the functionality to execute commands on the victim's machine. A notable difference of this version is its ability to create scheduled tasks using the Golang [Go-ole](#) library. This library leverages Windows Component Object Model (COM) object interfaces for interacting with the Task Scheduler service.



```
v265 = a3;
Object = github_com_go_ole_go_ole_oleutil_CreateObject(
    (unsigned int)"Schedule.Service",
    16,
    a3,
    n_6__2,
    a5,
    a6,
    a7,
    a8,
    a9,
    v126,
    a1a);
v254[0] = MEMORY[0x18];
v254[1] = v11;
v12 = qword_26E4B605478;
n_6__2 = 1LL;
a5 = 1LL;
fmt_Fprintln(
    (unsigned int)off_26E4B450B20
```

CR4T using go-ole library

The malware is also capable of achieving persistence by utilizing the [COM objects hijacking](#) technique. And finally, it uses the Telegram API for C2 communications, implementing the public [Golang Telegram API](#) bindings. All the interactions are similar to the C/C++ version.

Infrastructure

The infrastructure used in this campaign appears to be located in the US at two different commercial hosters.

Domain	IP	First seen	ASN
commonline[.]space	135.148.113[.]161	2023-12-16 23:20	16276
userfeedsync[.]com	104.36.229[.]249	2024-01-10 07:27	395092

Victims

We discovered victims in the Middle East, as per our telemetry, as early as February 2023. Additionally, there were several uploads to a semi-public malware scanning service at a later stage, more specifically starting on December 12 2023, with more than 30 submissions of the droppers in the period up to the end of January 2024. The majority of these uploads also originated from the Middle East. Other sources we suspect to be VPN exit nodes geo-located in South Korea, Luxembourg, Japan, Canada, Netherlands and the US.

Conclusions

The “DuneQuixote” campaign targets entities in the Middle East with an interesting array of tools designed for stealth and persistence. Through the deployment of memory-only implants and droppers masquerading as legitimate software, mimicking the Total Commander installer, the attackers demonstrate above average evasion capabilities and techniques. The discovery of both C/C++ and Golang versions of the CR4T implant highlights the adaptability and resourcefulness of the threat actors behind this campaign.

Indicators of Compromise

DuneQuixote Droppers

3aaf7f7f0a42a1cf0a0f6c61511978d7
5759acc816274d38407038c091e56a5c
606fdee74ad70f76618007d299adb0a4
5a04d9067b8cb6bcb916b59dcf53bed3
48c8e8cc189eef04a55ecb021f9e6111
7b9e85afa89670f46f884bb3bce262b0
4f29f977e786b2f7f483b47840b9c19d
9d20cc7a02121b515fd8f16b576624ef
4324cb72875d8a62a210690221cdc3f9
3cc77c18b4d1629b7658afb4175222c
6cfec4bdcbcf7f99535ee61a0ebae5dc
c70763510953149fb33d06bef160821c
f3988b8aaaa8c6a9ec407cf5854b0e3b
cf4bef8537c6397ba07de7629735eb4e
1bba771b9a32f0aada6eae64643673a
72c4d9bc1b59da634949c555b2a594b1
cc05c7bef5cff67bc74fda2fc96ddf7b
0fdbe82d2c8d52ac912d698bb8b25abc
9b991229fe1f5d8ec6543b1e5ae9beb4
5e85dc7c6969ce2270a06184a8c8e1da
71a8b4b8d9861bf9ac6bd4b0a60c3366
828335d067b27444198365fac30aa6be
84ae9222c86290bf585851191007ba23
450e589680e812ffb732f7e889676385
56d5589e0d6413575381b1f3c96aa245
258b7f20db8b927087d74a9d6214919b
a4011d2e4d3d9f9fe210448dd19c9d9a
b0e19a9fd168af2f7f6cf997992b1809
0d740972c3dff09c13a5193d19423da1
a0802a787537de1811a81d9182be9e7c
5200fa68b6d40bb60d4f097b895516f0
abf16e31deb669017e10e2cb8cc144c8
f151be4e882352ec42a336ca6bff7e3d
f1b6aa55ba3bb645d3fde78abda984f3
00130e1e7d628c8b5e2f9904ca959cd7
fb2b916e44abddd943015787f6a8dc35
996c4f78a13a8831742e86c052f19c20
4f29f977e786b2f7f483b47840b9c19d
91472c23ef5e8b0f8dda5fa9ae9afa94
135abd6f35721298cc656a29492be255
db786b773cd75483a122b72fdc392af6

Domains and IPs

Commonline[.]space
g1sea23g.commonline[.]space
tg1sea23g.commonline[.]space
telemetry.commonline[.]space
e1awq1lp.commonline[.]space

mc.commonline[.]space
userfeedsync[.]com
Service.userfeedsync[.]com
telemetry.userfeedsync[.]com