Tracking a P2P network related to TA505

research.nccgroup.com/2021/12/01/tracking-a-p2p-network-related-with-ta505/

December 1, 2021

This post is by **Nikolaos Pantazopoulos** and **Michael Sandee**

tl;dr – Executive Summary

For the past few months NCC Group has been closely tracking the operations of TA505 and the development of their various projects (e.g. Clop). During this research we encountered a number of binary files that we have attributed to the developer(s) of 'Grace' (i.e. FlawedGrace). These included a remote administration tool (RAT) used exclusively by TA505. The identified binary files are capable of communicating with each other through a peer-to-peer (P2P) network via UDP. While there does not appear to be a direct interaction between the identified samples and a host infected by 'Grace', we believe with medium to high confidence that there is a connection to the developer(s) of 'Grace' and the identified binaries.

In summary, we found the following:

- P2P binary files, which are downloaded along with other Necurs components (signed drivers, block lists)
- P2P binary files, which transfer certain information (records) between nodes
- Based on the network IDs of the identified samples, there seem to be at least three different networks running
- The programming style and dropped file formats match the development standards of 'Grace'

History of TA505's Shift to Ransomware Operations

2014: Emergence as a group

The threat actor, often referred to as TA505 publicly, has been distinguished as an independent threat actor by NCC Group since 2014. Internally we used the name "Dridex RAT group". Initially it was a group that integrated quite closely with EvilCorp, utilising their Dridex banking malware platform to execute relatively advanced attacks, using often custom made tools for a single purpose and repurposing commonly available tools such as 'Ammyy Admin' and 'RMS'/'RUT' to complement their arsenal. The attacks performed mostly consisted of compromising organisations and social engineering victims to execute high value bank transfers to corporate mule accounts. These operations included social engineering correctly implemented two-factor authentication with dual authorization by both the creator of a transaction and the authorizee.

2017: Evolution

Late 2017, EvilCorp and TA505 (Dridex RAT Group) split as a partnership. Our hypothesis is that EvilCorp had started to use the Bitpaymer ransomware to extort organisations rather than doing banking fraud. This built on the fact they had already been using the Locky ransomware previously and was attracting unwanted attention. EvilCorp's ability to execute enterprise ransomware across large-scale businesses was first demonstrated in May 2017. Their capability and success at pulling off such attacks stemmed from the numerous years of experience in compromising corporate networks for banking fraud activity, specifically moving laterally to separate hosts controlled by employees who had the required access and control of corporate bank accounts. The same techniques in relation to lateral movement and tools (such as Empire, Armitage, Cobalt Strike and Metasploit) enabled EvilCorp to become highly effective in targeted ransomware attacks.

However in 2017 TA505 went on their own path and specifically in 2018 executed a large number of attacks using the tool called 'Grace', also known publicly as 'FlawedGrace' and 'GraceWire'. The victims were mostly financial institutions and a large number of the victims were located in Africa, South Asia, and South East Asia with confirmed fraudulent wire transactions and card data theft originating from victims of TA505. The tool 'Grace' had some interesting features, and showed some indications that it was originally designed as banking malware which had latterly been repurposed. However, the tool was developed and was used in hundreds of victims worldwide, while remaining relatively unknown to the wider public in its first years of use.

2019: Clop and wider tooling

In early 2019, TA505 started to utilise the Clop ransomware, alongside other tools such as 'SDBBot' and 'ServHelper', while continuing to use 'Grace' up to and including 2021. Today it appears that the group has realised the potential of ransomware operations as a viable business model and the relative ease with which they can extort large sums of money from victims.

The remainder of this post dives deeper into a tool discovered by NCC Group that we believe is related to TA505 and the developer of 'Grace'. We assess that the identified tool is part of a bigger network, possibly related with Grace infections.

Technical Analysis

The technical analysis we provide below focuses on three components of the execution chain:

1. A downloader – Runs as a service (each identified variant has a different name) and downloads the rest of the components along with a target processes/services list that the driver uses while filtering information. Necurs have used similar downloaders in the past.

2. A signed driver (both x86 and x64 available) – Filters processes/services in order to avoid detection and/or prevent removal. In addition, it injects the payload into a new process.
3. Node tool – Communicates with other nodes in order to transfer victim's data.

It should be noted that for all above components, different variations were identified. However, the core functionality and purposes remain the same.

Upon execution, the downloader generates a GUID (used as a bot ID) and stores it in the `ProgramData` folder under the filename `regid.1991-06.com.microsoft.dat`. Any downloaded file is stored temporarily in this directory. In addition, the downloader reads the version of `crypt32.dll` in order to determine the version of the operating system.

Next, it contacts the command and control server and downloads the following files:

- t.dat – Expected to contain the string 'kwREgu73245Nwg7842h'
- p3.dat – P2P Binary. Saved as 'payload.dll'
- d1c.dat – x86 (signed) Driver
- d2c.dat – x64 (signed) Driver
- bn.dat – List of processes for the driver to filter. Stored as 'blacknames.txt'
- bs.dat – List of services' name for the driver to filter. Stored as 'blacksigns.txt'
- bv.dat – List of files' version names for the driver to filter. Stored as 'blackvers.txt'.
- r.dat – List of registry keys for the driver to filter. Stored as 'registry.txt'

The network communication of the downloader is simple. Firstly, it sends a GET request to the command and control server, downloads and saves on disk the appropriate component. Then, it reads the component from disk and decrypts it (using the RC4 algorithm) with the hardcoded key 'ABCDF343fderfds21'. After decrypting it, the downloader deletes the file.

Depending on the component type, the downloader stores each of them differently. Any configurations (e.g. list of processes to filter) are stored in registry under the key *`HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID`* with the value name being the thread ID of the downloader. The data are stored in plaintext with a unique ID value at the start (e.g. 0x20 for the processes list), which is used later by the driver as a communication method.

In addition, in one variant, we detected a reporting mechanism to the command and control server for each step taken. This involves sending a GET request, which includes the generated bot ID along with a status code. The below table summarises each identified request (Table 1).

Request	**Description**
/c/p1/dnsc.php?n=%s&in=%s	First parameter is the bot ID and the second is the formatted string ("Version_is_%d.%d_(%d)_%d__ARCH_%d"), which contains operating system info
/c/p1/dnsc.php?n=%s&sz=DS_%d	First parameter is the bot ID and the second is the downloaded driver's size
/c/p1/dnsc.php?n=%s&er=ERR_%d	First parameter is the bot ID and the second is the error code
/c/p1/dnsc.php?n=%s&c1=1	The first parameter is the bot ID. Notifies the server that the driver was installed successfully
/c/p1/dnsc.php?n=%s&c1=1&er=REB_ERR_%d	First parameter is the bot ID and the second is the error code obtained while attempting to shut down the host after finding Windows Defender running
/c/p1/dnsc.php?n=%s&sz=ErrList_%d_%	First parameter is the bot ID, second parameter is the resulted error code while retrieving the blocklist processes. The third parameter is set to 1. The same command is also issued after downloading the blacklisted services' names and versions. The only difference is on the third parameter, which is increased to '2' for blacklisted services, '3' for versions and '4' for blacklisted registry keys
/c/p1/dnsc.php?n=%s&er=PING_ERR_%d	First parameter is the bot ID and the second parameter is the error code obtained during the driver download process
/c/p1/dnsc.php?n=%s&c1=1&c2=1	First parameter is the bot ID. Informs the server that the bot is about to start the downloading process.
/c/p1/dnsc.php?n=%s&c1=1&c2=1&c3=1	First parameter is the bot ID. Notified the server that the payload (node tool) was downloaded and stored successfully

Table 1 – Reporting to C2 requests

Driver Analysis

The downloaded driver is the same one that Necurs uses. It has been analysed publically already [1] but in summary, it does the following.

In the first stage, the driver decrypts shellcode, copies it to a new allocated pool and then executes the payload. Next, the shellcode decrypts and runs (in memory) another driver (stored encrypted in the original file). The decryption algorithm remains the same in both cases:

```
xor_key =  extracted_xor_key
bits = 15
result = b''
for i in range(0,payload_size,4):
        data = encrypted[i:i+4]
        value = int.from_bytes (data, 'little' )^ xor_key
        result += ( _rol(value, bits, 32)  ^ xor_key).to_bytes(4,'little')
```

Eventually, the decrypted driver injects the payload (the P2P binary) into a new process (‘wmiprvse.exe’) and proceeds with the filtering of data.

A notable piece of code of the driver is the strings’ decryption routine, which is also present in recent GraceRAT samples, including the same XOR key (*1220A51676E779BD877CBECAC4B9B8696D1A93F32B743A3E6790E40D745693DE58B1DD17F65988BEFE1D6C62D5416B25BB78EF0*

Payload Attribution and Analysis

The identified sample is written in C++ and interacts with other nodes in the network using UDP. We believe that the downloaded binary file is related with TA505 for (at least) the following reasons:

1. Same serialisation library
2. Same programming style with ‘Grace’ samples
3. Similar naming convention in the configuration’s keys with ‘Grace’ samples
4. Same output files (dsx), which we have seen in previous TA505 compromises. DSX files have been used by ‘Grace’ operators to store information related with compromised machines.

Initialisation Phase

In the initialisation phase, the sample ensures that the configurations have been loaded and the appropriate folders are created.

All identified samples store their configurations in a resource with name `XC`.

ANALYST NOTE: *Due to limit visibility of other nodes, we were not able to identify the purpose of each key of the configurations.*

The first configuration stores the following settings:

- cx – Parent name
- nid – Node ID. This is used as a network identification method during network communication. If the incoming network packet does not have the same ID then the packet is treated as a packet from a different network and is ignored.
- dgx – Unknown
- exe – Binary mode flag (DLL/EXE)
- key – RSA key to use for verifying a record
- port – UDP port to listen
- va – Parent name. It includes the node IPs to contact.

The second configuration contains the following settings (or metadata as the developer names them):

- meta – Parent name
- app – Unknown. Probably specifies the variant type of the server. The following seem to be supported:
  target (this is the current set value)
  gate
  drop
  control
- mod – Specifies if current binary is the core module.
- bld – Unknown
- api – Unknown
- llr – Unknown
- llt- Unknown

Next, the sample creates a set of folders and files in a directory named ‘target’. These folders are:

- node (folder) – Stores records of other nodes
- trash (folder) – Move files for deletion
- units (folder) – Unknown. Appears to contain PE files, which the core module loads.
- sessions (folder) – Active nodes’ sessions
- units.dsx (file) – List of ‘units’ to load
- probes.dsx (file) – Stores the connected nodes IPs along with other metadata (e.g. connection timestamp, port number)
- net.dsx (file) – Node peer name
- reports.dsx (file) – Used in recent versions only. Unknown purpose.

Network communication

After the initialisation phase has been completed, the sample starts sending UDP requests to a list of IPs in order to register itself into the network and then exchange information.

Every network packet has a header, which has the below structure:

```
struct Node_Network_Packet_Header
{
 BYTE XOR_Key;
 BYTE Version; // set to 0x37 ('7')
 BYTE Encrypted_node_ID[16]; // XORed with XOR_Key above
 BYTE Peer_Name[16]; // Xored with XOR_Key above. Connected peer name
 BYTE Command_ID; //Internally called frame type
 DWORD Watermark; //XORed with XOR_Key above
 DWORD Crc32_Data; //CRC32 of above data
};
```

When the sample requires adding additional information in a network packet, it uses the below structure:

```
struct Node_Network_Packet_Payload
{
 DWORD Size;
 DWORD CRC32_Data;
 BYTE Data[Size]; // Xored with same key used in the header packet (XOR_Key)
};
```

As expected, each network command (Table 2) adds a different set of information in the 'Data' field of the above structure but most of the commands follow a similar format. For example, an 'invitation' request (Command ID 1) has the structure:

```
struct Node_Network_Invitation_Packet
{
 BYTE CMD_ID;
 DWORD Session_Label;
 BYTE Invitation_ID[16];
 BYTE Node_Peer_Name[16];
 WORD Node_Binded_Port;
};
```

The sample supports a limited set of commands, which have as a primary role to exchange 'records' between each other.

Command ID	Description
1	Requests to register in the other nodes ('invitation' request)
2	Adds node IP to the probes list
3	Sends a ping request. It includes number of active connections and records
4	Sends number of active connections and records in the node
5	Adds a new node IP:Port that the remote node will check
6	Sends a record ID along with the number of data blocks
7	Requests metadata of a record
8	Sends metadata of a record
9	Requests the data of a record
10	Receives data of a record and store them on disk

Table 2 – Set of command IDs

ANALYST NOTE: *When information, such as record IDs or number of active connections/records, is sent, the binary adds the length of the data followed by the actual data. For example, in case of sending number of active connections and records:*

01 05 01 02 01 02

The above is translated as:

2 active connections from a total of 5 with 2 records.

Moreover, when a node receives a request, it sends an echo reply (includes the same packet header) to acknowledge that the request was read. In general, the following types are supported:

- Request type of 0x10 for echo request.
- Request type of 0x07 when sending data, which fit in one packet.
- Request type of 0xD when sending data in multiple packets (size of payload over 1419 bytes).
- Request type 0x21. It exists in the binary but not supported during the network communications.

Record files

As mentioned already, a record has its own sub-folder under the 'node' folder with each sub-folder containing the below files:

- m – Metadata of record file

- l – Unknown purpose
- p – Payload data

The metadata file contains a set of information for the record such as the node peer name and the node network ID. Among this information, the keys ‘tag’ and ‘pwd’ appear to be very important too. The ‘tag’ key represents a command (different from table 2 set) that the node will execute once it receives the record. Currently, the binary only supports the command ‘updates’. The payload file (p) keeps the updated content encrypted with the value of key ‘pwd’ being the AES key.

Even though we have not been able yet to capture any network traffic for the above command, we believe that it is used to update the current running core module.

IoCs

Nodes’ IPs

45.142.213[.]139:555

195.123.246[.]14:555

45.129.137[.]237:33964

78.128.112[.]139:33964

145.239.85[.]6:3333

Binaries

SHA-1	Description
A21D19EB9A90C6B579BCE8017769F6F58F9DADB1	P2P Binary
2F60DE5091AB3A0CE5C8F1A27526EFBA2AD9A5A7	P2P Binary
2D694840C0159387482DC9D7E59217CF1E365027	P2P Binary
02FFD81484BB92B5689A39ABD2A34D833D655266	x86 Driver
B4A9ABCAAADD80F0584C79939E79F07CBDD49657	x64 Driver
00B5EBE5E747A842DEC9B3F14F4751452628F1FE	X64 Driver
22F8704B74CE493C01E61EF31A9E177185852437	Downloader
D1B36C9631BCB391BC97A507A92BCE90F687440A	Downloader

Table 3 – Binaries hashes

References

1. https://pro-cdo-web-resources.s3.eu-west-1.amazonaws.com/elevenpaths/uploads/2020/6/elevenpaths-informe-aptualizator-2-en.pdf