The Brothers Grim

blog.group-ib.com/grimagent

02.07.2021

The reversing tale of GrimAgent malware used by Ryuk

Albert Priego

Malware Analyst at Group-IB

Executive summary

Ransomware activity increased drastically over the past couple of years and became the face of cybercrime by 2021.

According to the "Ransomware Uncovered 2020-2021" report, the number of ransomware attacks increased by more than 150% in 2020. The attacks grew in not only number but also scale and sophistication — the average ransom demand increased by more than twofold and amounted to $170,000 in 2020. The norm is shifting toward the millions: the Colonial Pipeline allegedly paid USD 5 million to get its business back. The case propelled the question of ransomware to the top of the political agenda.

In the meantime, 2021 continues to prove that no company is immune to the ransomware plague. Ransomware operators are not concerned about the industry so long as the victim can pay the ransom. The prospect of quick profits motivates new players to join big game hunting. Ransomware operations show no signs of slowing down. The gangs evolve. They change their tactics, defense evasion techniques, and procedures to ensure that their illicit business thrives.

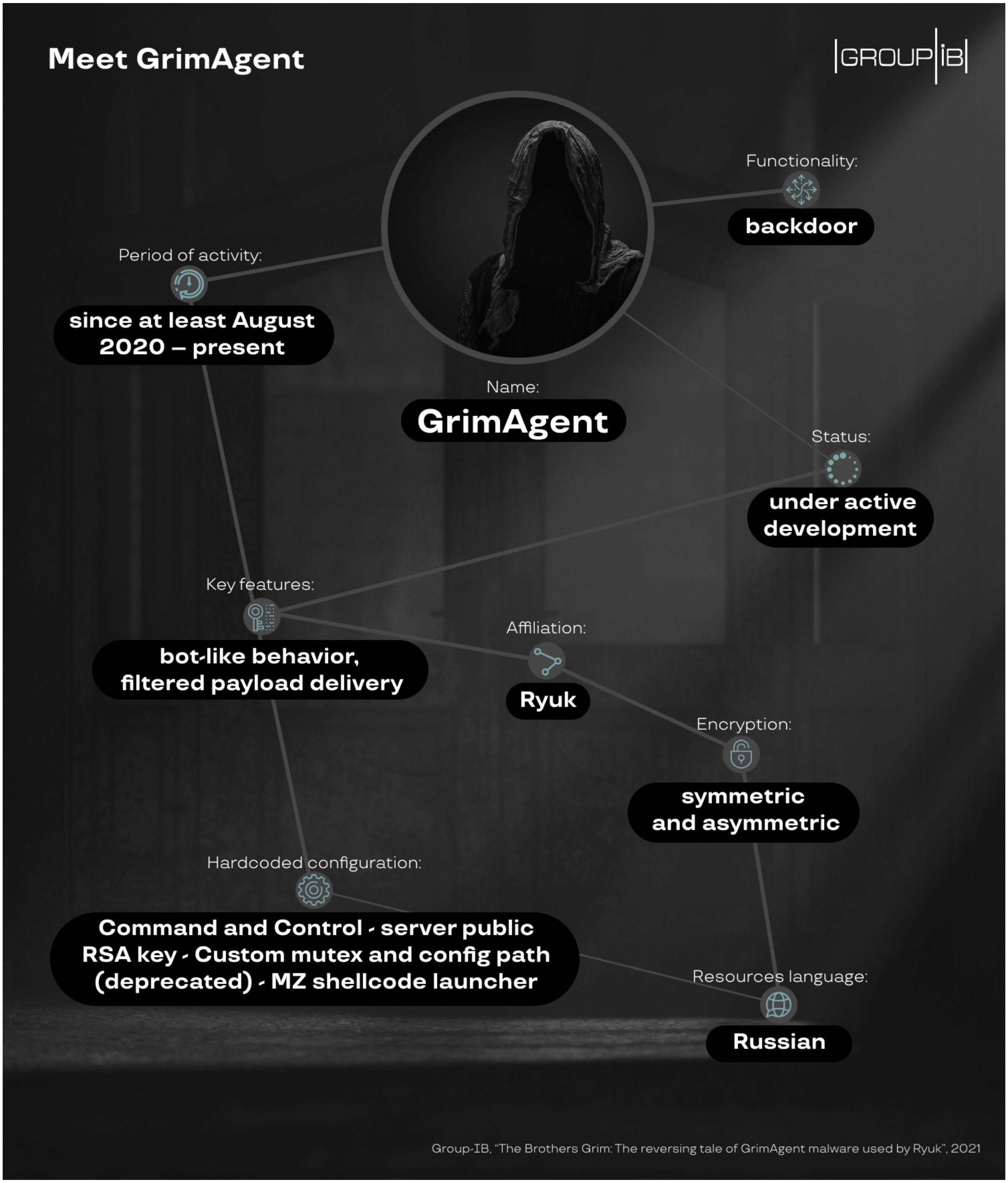
Given that ransomware attacks are conducted by humans, understanding the modus operandi and toolset used by attackers is essential for companies that want to avoid costly downtimes. Ultimately, knowing how ransomware gangs operate and being able to thwart their attacks is more cost-effective than paying ransoms.

One of the underlying trends of 2021 to keep in mind is the use of commodity malware. The infamous ransomware gang Ryuk, which is responsible for many high-profile cyber heists (including the attack on the Baltimore County Public Schools system) followed suit. The most recent addition to their arsenal, which is yet to be explored, is the malware called **GrimAgent**.

Our team did the first comprehensive analysis of the **GrimAgent backdoor**. It is intended mainly for reverse engineers, researchers and blue teams so that they can create and implement rules that help monitor this cyber threat closely. **Group-IB's Threat Intelligence team** has created Yara and Suricata rules as well as mapped GrimAgent's TTPs according to the **MITRE ATT&CK® matrix.**
You are reading a condensed summary of the actual reverse engineering of GrimAgent.
If you feel up to reading the full text of the report "The Brothers Grim: The reversing tale of GrimAgent malware used by Ryuk" click here.
Approximate reading time: ∞
Introduction

GrimAgent is a malware classified as a backdoor and that has been used as a prior stage to Ryuk ransomware. The ransomware family appeared in 2018 and was mistakenly linked to

North Korea. Later on, it was attributed to two threat actors, **FIN6** and **Wizard Spider**.

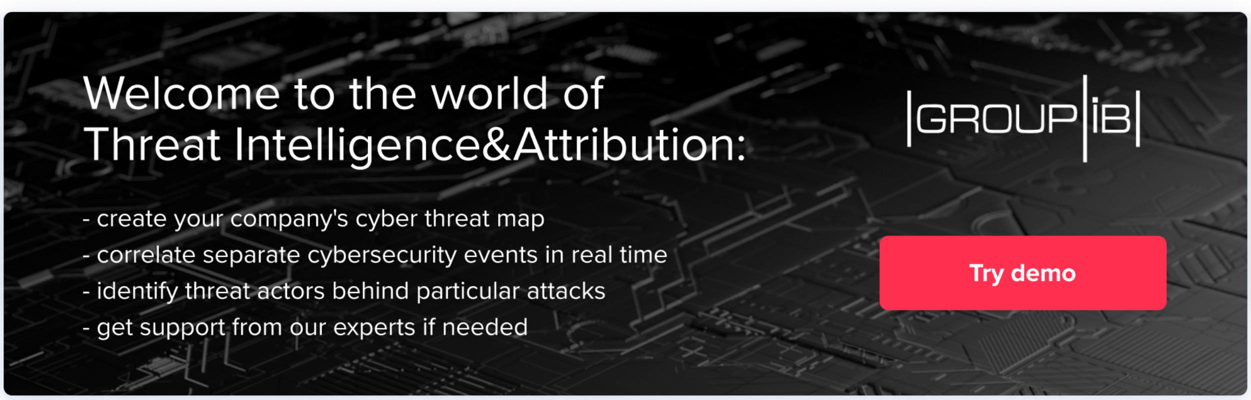


Given the limited knowledge about the links between Ryuk and GrimAgent, we decided to research GrimAgent samples discovered in the wild and show how GrimAgent is **connected to Ryuk**. The article analyzes the execution chain, TTPs, and the malware's relevant characteristics.

The first known GrimAgent sample (*SHA-256: 03ec9000b493c716e5aea4876a2fc5360c695b25d6f4cd24f60288c6a1c65ce5)* was **uploaded to VirusTotal on August 9, 2020 at 19:20:54.** It is noteworthy that **an embedded binary** into the initial malware was employed and had a **timestamp of 2020-07-26**, the timestamp could have been altered but the **dates coincide with our hypothesis** about the new malware.

From a functionality point of view the malware is a backdoor, but it behaves like a bot. We analyzed a completely different custom network protocol where the infected computer would register on the server side and provide a reconnaissance string of the client, after which it would constantly make requests to the C&C server asking which are the next commands to be executed. During our research we performed several tests with the aim to get the next stage payload. We infected several testing devices with different settings, but did not manage to obtain any payloads. Based on our findings, it is likely that the actor implemented different defense and delivery mechanisms to protect the integrity of its systems and ensure that the operations are flawless — which is not uncommon and we have witnessed this in the past. This means that GrimAgent developers potentially implemented threat detection systems capable of detecting sandboxes or bot requests in order to protect themselves from things such as analysis, added filters based on geolocation and blacklists/whitelists. The extreme meticulousness shown by the actors behind the malware and their attention to detail when carrying out attacks is both relevant and remarkable.

According to Group-IB's Threat Intelligence & Attribution system, Ryuk operators used different commodity malware over time (including Emotet, TrickBot, Bazar, Buer, and SilentNight) to deploy ransomware. However, the big blows suffered by Trickbot and Emotet could have prompted Ryuk operators to partner with GrimAgent. A detailed analysis of the latest TTPs used by various ransomware strains is available in the Ransomware Uncovered 2020/2021 report.



Connections to Ryuk

Analyzing the GrimAgent Command and Control domain revealed an interesting URL. When making a request on the domain, the Command and Control server returns a content designed for victims of the Ryuk ransomware, in addition to revealing its location on the TOR network.

Command and control landing page:

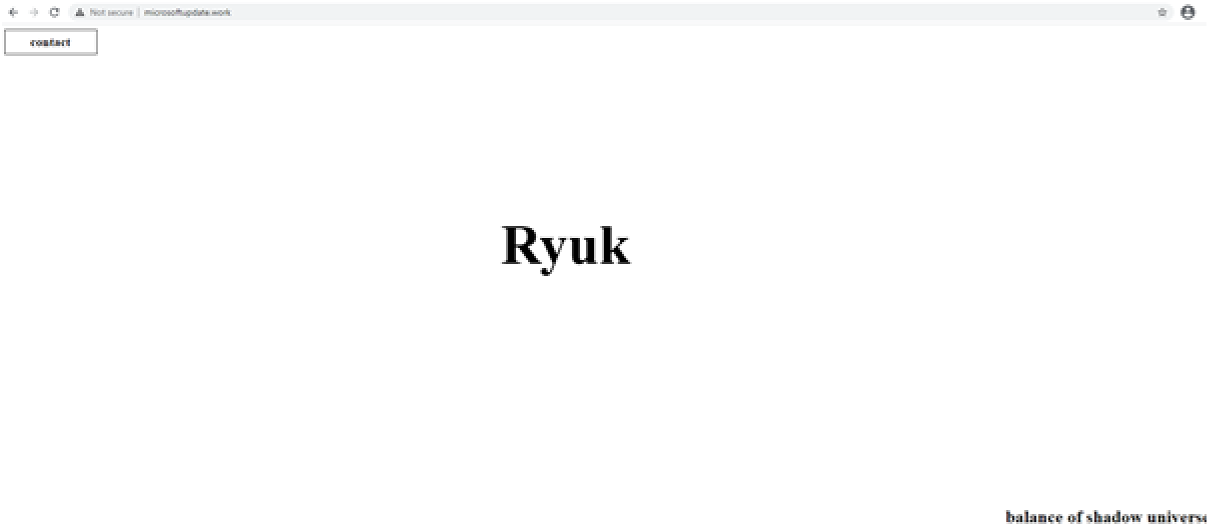


Fig. 1: C2 landing page

Command and control landing page source code:

```
< html > < body > < style > p:hover {
    background: black;
    color:white
} < /style><script> = 'IDyIOJ4u'; =
'http[:]//etnbhivw5fjqytbmvt2o6zle3avqn6rrugfc35kmcmedbbgqbxtknlqd[.]onion';
</script > < p onclick = 'info()' style = 'font-weight:bold;font-size:127%;
top:0;left:0;border: 1px solid black;padding: 7px 29px;width:85px;' >  
 
contact < /p><p style='position:absolute;bottom:0;right:1%;font-weight:bold;font-
size:171%'>&#98;&#97;&#108;&#97;&#110;&#99;&#101;&#32;&#111;&#102;&#32;&#115;&#104;&#9
</p > < div style = 'font-size: 551%;font-
weight:bold;width:51%;height:51%;overflow:auto;margin:auto;position:absolute;top:36%;l
 > &#82;
&#121;
&#117;
&#107; < /div><script>function info(){alert('INSTRUCTION:
1. Download tor browser.
2. Open link through tor browser: ' + + '
3. Fill the form, your password: '+ +'
We will contact you shortly.
Always send files for test decryption.');}; </script > < /body></html >
```

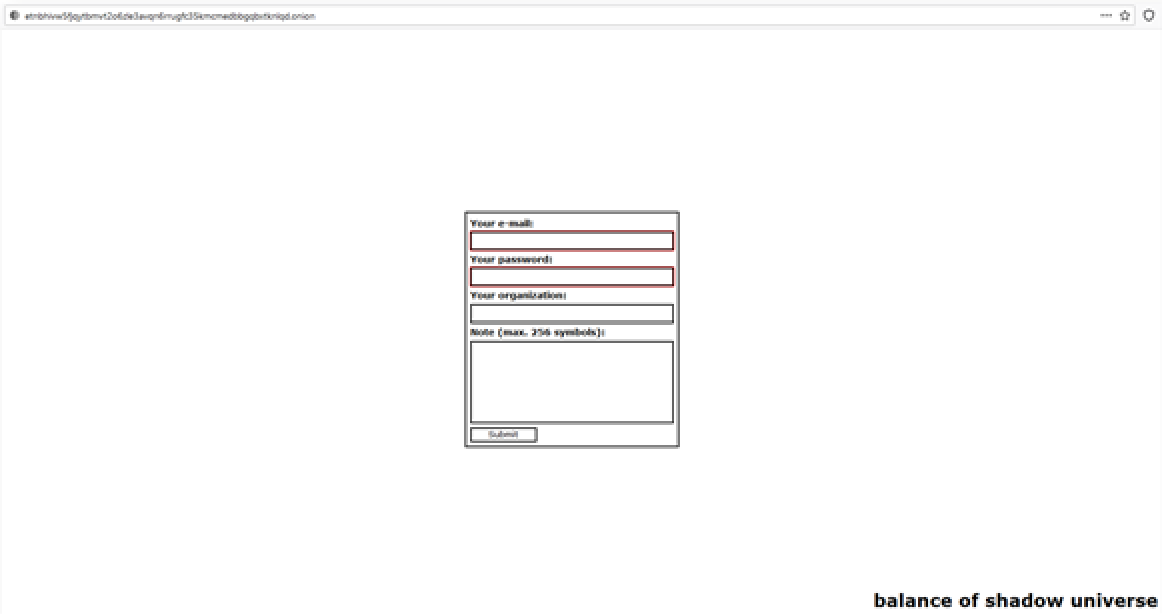


Fig. 2: Ryuk Cpanel (TOR)

These findings show that the **GrimAgent backdoor is being used as a part of Ryuk's operations**. We have not observed any selling or advertising related to GrimAgent on underground forums, nor any use of the malware in the infection chain of any other malware besides Ryuk. Based on the above facts we believe that this malware is used by the same TA that uses Ruyk ransomware and does not use MaaS.

GrimAgent in a nutshell

File Information

Functionality

- Retrieve information about the victim:
  - IP and country code
  - Domain
  - Vendor
  - Build version
  - OS
  - Arch
  - Username
  - Privileges

  - user_id (computed to identify infected clients)

- AES key obtained from C2 to encrypt the communications
- Execute
- Execute shellcode (MZ launcher)
- Download and execute
- Update
- Execute DLL (MZ launcher trampoline)

Encryption/decryption scheme

Two decryption keys used to decrypt the strings (decrypted by the same algorithm).

Four keys are used:

1. **Hardcoded server public RSA key:** Used to encrypt the first request to the Command and Control server and register the infected client on the server side.

2. **Generated client RSA public and private keys:** Both stored at the config file.

   a. Public RSA key
   i. Used to compute the user_id in order to track and identify infected users.
   ii. Sent to C2 on the first request along with client reconnaissance; used to encrypt the AES key sent by C2.

3. Used to compute the user_id in order to track and identify infected users.

   b. Private RSA key: Used to decrypt the AES key received.

4. **AES key:** Used on command and control further communications (symmetric encryption).

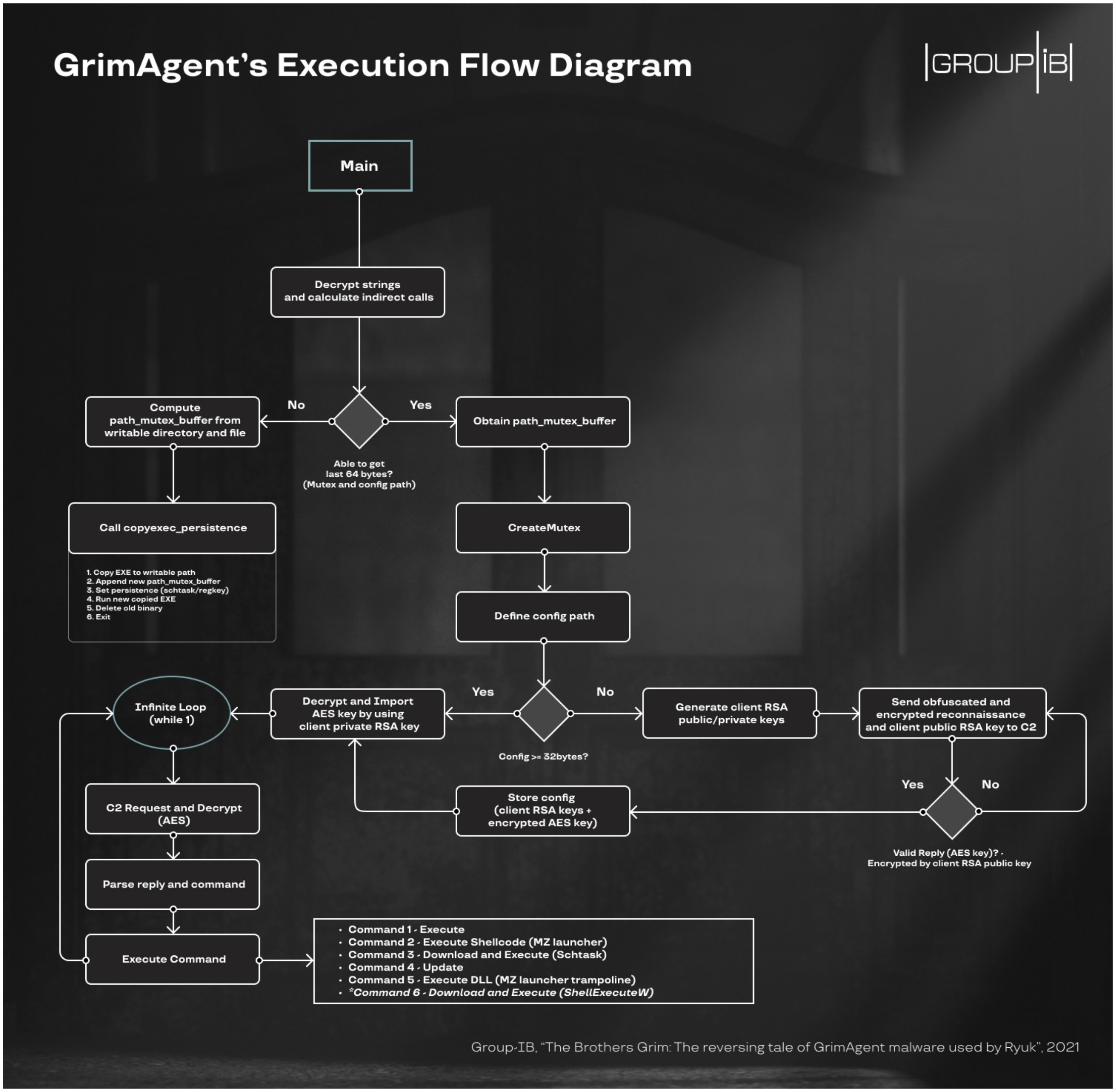


Fig. 3: Execution flow diagram

**Command 6 has been added on new versions of the malware, check "Dealing with newer GrimAgent versions" section.*

In-depth analysis

This section details the malware's behavior. It covers the following points:

- Bypassing anti-analysis mechanisms
- Malware execution
- Network protocol

Bypassing anti-analysis mechanisms

Overlapping instruction obfuscation

Our investigation involved dealing with anti-analysis techniques, which complicated our task of analyzing the malware in addition to breaking the IDAPro Hex-Rays display view.

Basically, the execution flow has been altered with the entire code region between the pusha and popa instructions, as well as the stack usage. This code region does not perform any relevant actions while executing the binary, so we can NOP all these instructions.

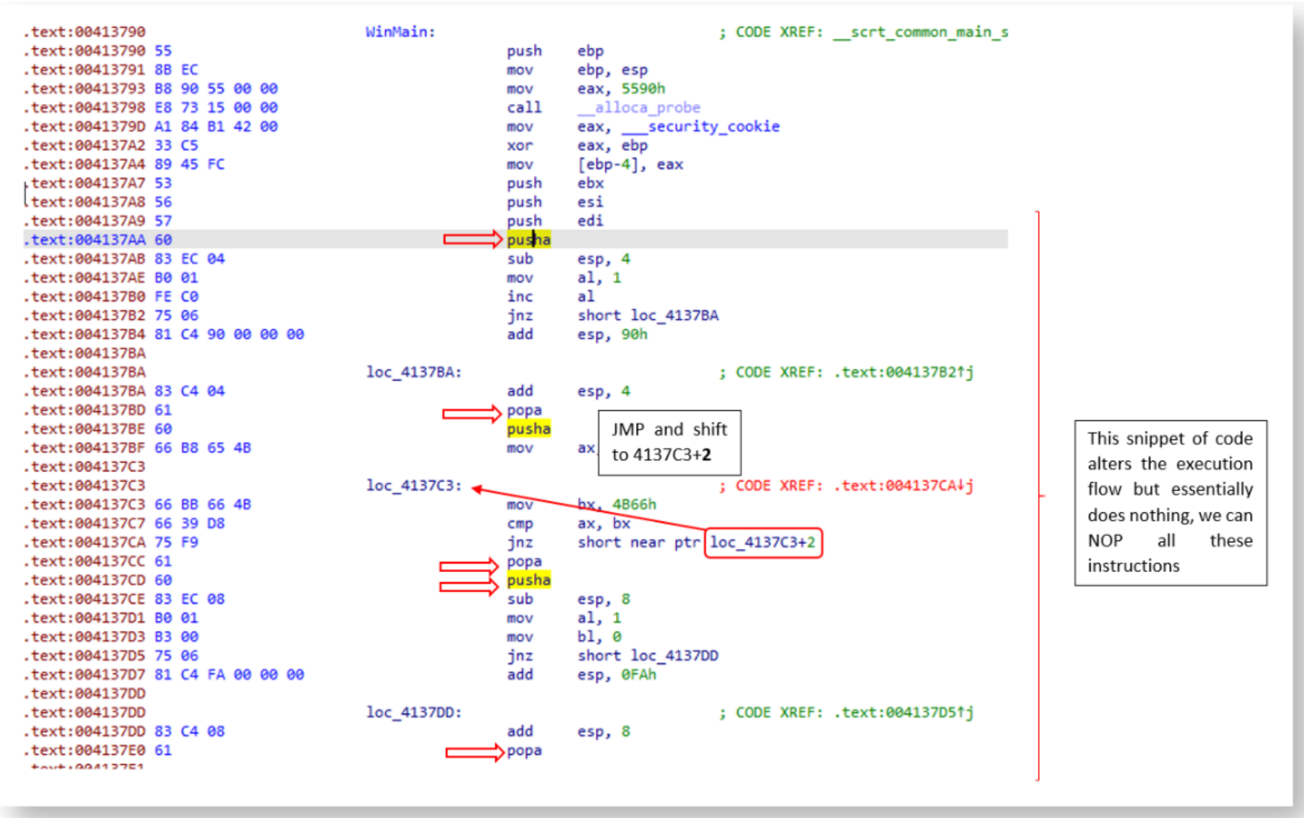


Fig. 4: Anti-analysis technique at WinMain

Once done, we can set WinMain as a function and try to visualize the pseudocode as well as the IDA graph mode.

String encryption

When executing the WinMain routine, the first step is to decrypt the key that will later be used to decrypt the sensitive strings needed to continue its execution. In order to decrypt the key, the malware uses a function which may vary depending on the sample.

Then, decrypt some basic strings to be able to execute their most basic actions and decrypt the second key that will be used on the decryption of all the necessary strings for their complete and correct execution.

IDA Python script

Since, in most cases, it is better to automate than perform a task manually, we use IDA Python to ensure it is done quickly.

IDAPro script:
https://github.com/apriegob/GrimAgent/blob/main/IDAPro/IDAPro_string_decryptor.py

The "*decrypt_strings_func*" and "*decrypt_strings2_func*" values should be changed to the memory location where they are placed. The value of key1 and key2 will be the buffers that contain the keys used by the malware (which are easily obtained through the debugger).
Malware execution

The execution of GrimAgent from can be divided into the following subsections.

- Decrypting strings and calculating indirect calls
- path_mutex_buffer
- Mutex
- Configuration file
- First launch of the malware
- Handling commands
- Dealing with newer GrimAgent versions

Decrypting strings and calculating indirect calls

This is the first step the malware must take because it needs to decrypt the sensitive and relevant strings as well as calculate the indirect calls in order to continue with its normal execution flow.

path_mutex_buffer (deprecated)

One of the most relevant features of GrimAgent is that it uses the last 64 bytes of the binary for two purposes:

1. **Compute mutex name**
2. **Check path for store malware configuration**

This means that after decrypting strings and calculating indirect calls, GrimAgent will read itself, and more specifically its last 64 bytes. Henceforth we will call this set of bytes "*path_mutex_buffer*".

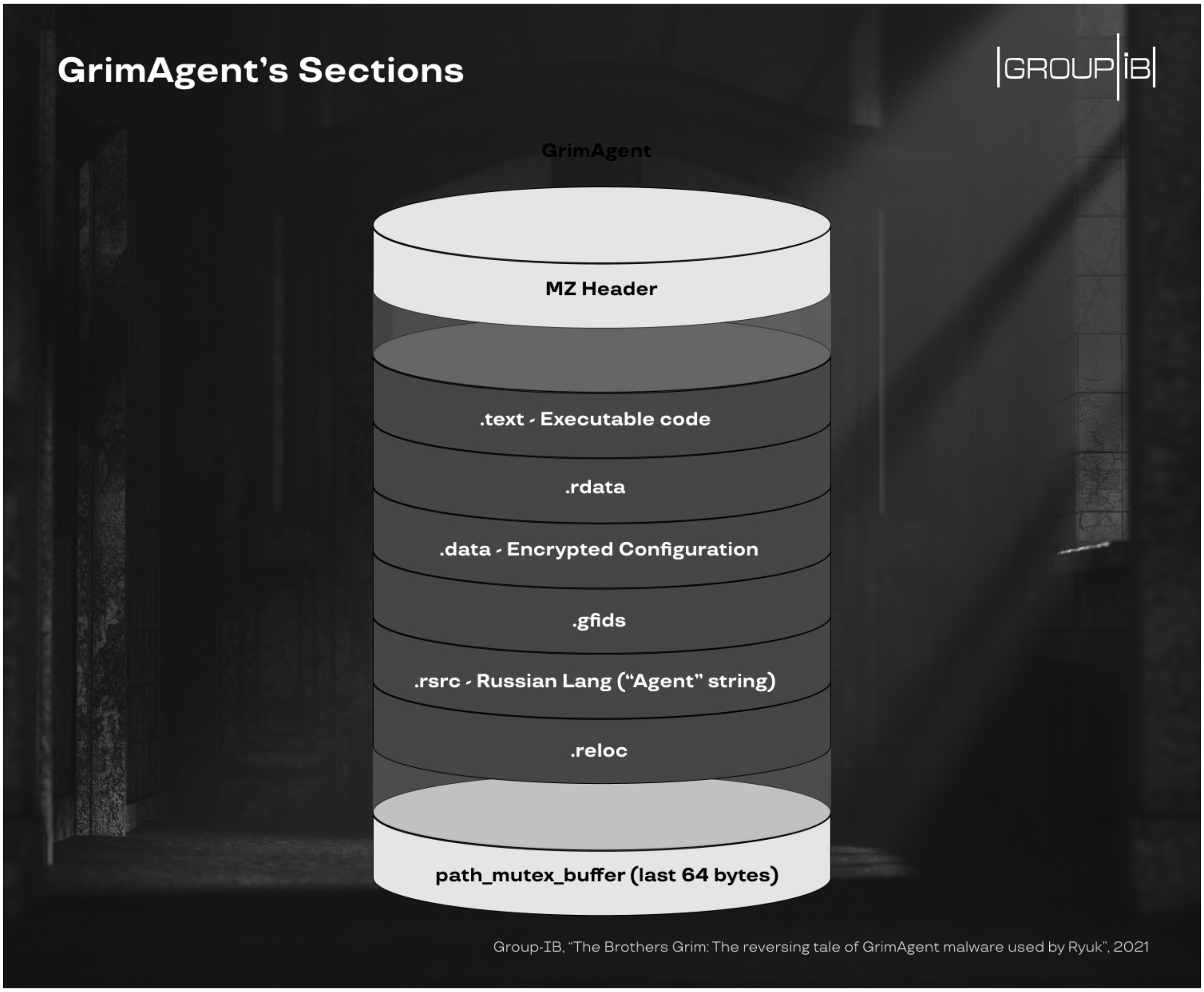


Fig. 5: GrimAgent sections

Mutex

The function that creates the mutex will iterate over the 64 bytes, checking if the value is alphanumeric and, if it is not, the function will try to transform it into an alphanumeric value.

If the name of the mutex created is not valid, it will create a mutex with the hardcoded name "*mymutex*". In case of an error, the execution will terminate.

```
int __cdecl mutex_creation(char *mutex_name_buff)
{
  int i; // [esp+1Ch] [ebp-50h]
  int iter; // [esp+20h] [ebp-4Ch]
  unsigned __int8 char_mx; // [esp+27h] [ebp-45h]
  unsigned __int8 int_mx; // [esp+27h] [ebp-45h]
  char mutex_name[64]; // [esp+28h] [ebp-44h] BYREF

  memset(mutex_name, 0, sizeof(mutex_name));
  iter = 0;
  for ( i = 0; i < 64; ++i )
  {
    char_mx = mutex_name_buff[i];
    if ( isalnum(char_mx) )
    {
      mutex_name[iter++] = char_mx;
    }
    else
    {
      int_mx = mutex_name_buff[i] % 74 + 48;
      if ( isalnum(int_mx) )
        mutex_name[iter++] = int_mx;
    }
  }
  if ( &mutex_name[strlen(mutex_name) + 1] == &mutex_name[1] )
    strcpy(mutex_name, "mymutex");
  if ( call_CreateMutexA(0, 0, mutex_name) && GetLastError() == 183 )
    _Exit((char *)1);
  return 1;
}
```

Iterates through the 64 bytes and compute the mutex name

Case of byte non-alphanumeric, then try to transform it to alphanumeric

Fig. 6: Computing Mutex

Configuration file

The malware will check if it has a specific path hardcoded. The path corresponds to the buffer that it read in its last 64 bytes ("*path_mutex_buffer*").

Custom path syntax (deprecated):

> SHA256 [unicode(FilePath)] + SHA256 [unicode(Filename)] → length of SHA256 x 2 = 64 (0x40) bytes

To do so, the malware will make **recursive calls to check if the SHA256 of the different writable paths and filenames matches** with the buffer retrieved.

In terms of writing its config, the malware has three possibilities:

1. **Custom Path** → SHA256[unicode(Path)] + SHA256[unicode(FileName)]: The malware will start from "*C:\"* path, and it will call a function recursively (find_custompath) in which it will compute the SHA256 of the different directories to see if it matches with the first 32 bytes of "*path_mutex_buffer*" and, if It does, it will try to find the filename by performing the SHA256 of every filename in the matched directory by checking the next 32 bytes.
2. **Hardcoded path 1** → *C:\Users\Public\config*; used if unable to get the custom path.
3. **Hardcoded path 2** → *C:\Users\Public\reserve.sys*

- a. This path will be used when a custom path is matched with the character ' * ', for example: "C:\Users*"
- b. There is a string "reserve.exe" that will replace the last 3 characters from "exe" to "sys" during execution time.

With the config path defined, the malware checks if there is a valid config file:

> **If the config file is less than 0x32 bytes in size, it has no valid config**. Otherwise, the config will be considered as valid and continue the execution to an infinite loop that will perform C2 requests asking for new commands to execute.

A valid configuration file will contain three different objects:

1. **Generated client public RSA key**: to compute user_id and encrypt the AES key that the C2 will send.
2. **Generated client private RSA key**: to decrypt the AES key sent by C2.
3. **AES encrypted key** received from C2: to encrypt/decrypt C2 communications.

First launch of the malware

When the malware is executed, the first step is to decrypt the strings and create a mutex based on the last 64 bytes (*path_mutex_buffer),* but there are cases where GrimAgent binaries have been compiled without these last bytes*.*

Unable to read its last 64 bytes

This execution branch can only occur on malware first execution. If we enter this execution branch, the malware will perform the following actions:

- Retrieve the writable path
- Compute the new *path_mutex_buffer* based writable path and filename
- Copy itself into the writable path
- Append the new *path_mutex_buffer* at the end of the new copied binary

- Set persistence of the copied binary
    - Task scheduler
    - Registry key
- Execute copied malware
- Delete old sample
- Exits

Reconnaissance

At this point, the malware will have been able to get its path to store its configuration file where it will store the keys necessary for its execution and communication with C2 as well as for creating a mutex to avoid conflicts if it is executed several times.

GrimAgent **will then perform a system reconnaissance and collect device information**. It will collect the following fields:

- Country code (api.myip.com)
- IP (api.myip.com)
- Vendor
- Domain
- Build Version
- OS
- Architecture
- Username
- Privileges (A/U)
    - A = Admin
    - U = User

To retrieve fields such as the internet IP address and country code, which will append into the reconnaissance string, the malware will use public resources. In our case, this means performing a GET request to api.myip.

RSA key generation and user_id

GrimAgent calls a function in order to **compute a user_identifer** (used to identify the infected user) **and generates the client RSA public and private keys**:

1. **Generates the client RSA public and private keys** by using CryptGenKey.

2. **Retrieves the new generated client RSA public key in PEM format** (with the headers '-----BEGIN PUBLIC KEY-----' and '-----END PUBLIC KEY-----').
3. **Computes the hash for that key, which generates a random id** to be included in the reconnaissance string (used as a user identifier): This user_id would be different for every execution but, to be able to identify the infected user and not generate a random key each time, the new key will be stored and used to compute the user_id when needed.
4. **Retrieves the generated client RSA private key**.

5. **Stores both public and private keys** in the config file.

The next screenshot shows the finished string obtained by the malware with the value of user_id added at the end.

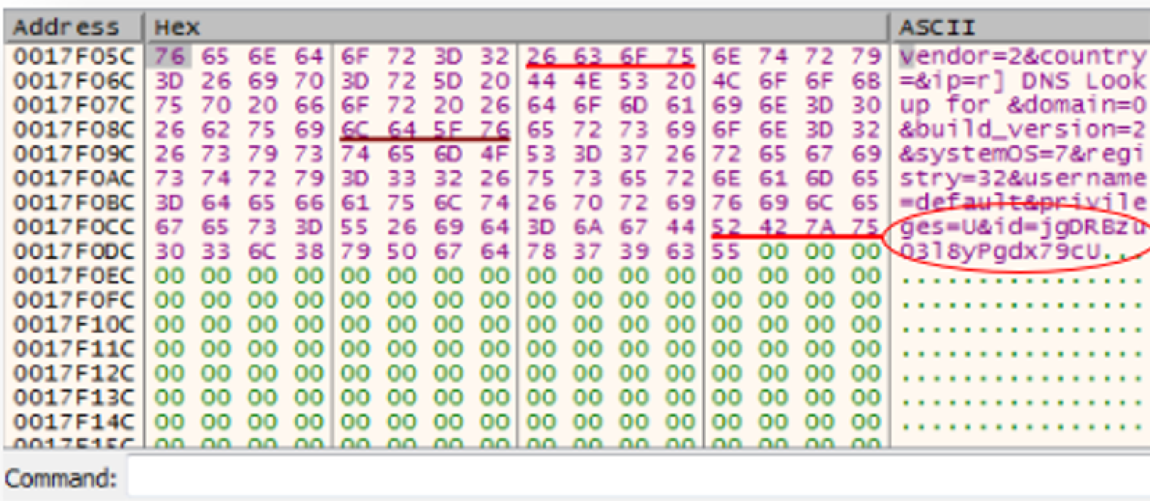


Fig. 7 - Complete victim reconnaissance
NB: Country code and IP address values are invalid because we executed the malware in a controlled environment without internet access.

Example of a reconnaissance string:

vendor=<vendor_id>&country=<country>&ip=<ip>&domain=<domain>&build_version=<build_version>&systemOS=<OS>&registry=<32/64>&username=<username>&privileges=<U/A>&id=<user_id>

Next, the malware **sends this information along with the generated client public RSA key** to the Command and Control server and waits until the reply contains 'fr' or 'ar' as the first characters in the reply. Only if the first characters received are 'fr' will the malware store the reply as hex values. **The reply from C2 will be a symmetric encryption key (AES) that the malware will use from then on in order to interact with the Command and Control server.**

The malware will JMP to the '*exist_conf*' location and proceed with the execution.

Handling commands

We land in this section once the malware obtains a valid configuration and is about to enter an infinite loop where it will make requests for new commands to the Command and Control server. For GrimAgent, the configuration file will be valid if it is ≥ 32 bytes.

Reading config: the AES Key

In this case, the malware will decrypt and import the AES key received from the C2 which will be in the config file. Then, the key will be used in order to encrypt and decrypt subsequent communications between the infected client and the C2.

Commands

GrimAgent will start an infinite loop (while 1) in order to request the next commands to execute to the Command and Control server. The loop is made up of three points:

1. C2 request and decryption
2. Execute command
3. Sleep [180-190] seconds and start again

Command table:

The commands and other malware features have been updated in newer versions of the malware, explained at Dealing with newer GrimAgent versions section.

On receiving the C2 reply, the malware parses the command and executes it.

Command 1: Execute

At this point, the malware uses the same logic for execution that we have already seen. It creates a task with a random name and length 8. The task will be executed through the task scheduler with maximum privileges. Afterwards, the malware removes the task to avoid being detected.

Command 2: Execute shellcode (MZ launcher)

When command '2' is received, the malware parses the shellcode received as an argument. The malware checks for '\\' and 'x' bytes, which are common delimiters used in binary data.

After parsing the shellcode, the malware calls the '*drop_and_execute_shellcode*' function, which **drops an executable MZ (embedded into initial binary) with the appended shellcode into a writable directory that will act as a launcher of the received binary data**. The embedded MZ contains the compiler stamp from July 2020. This point in time could be when the first version of GrimAgent appeared. Moreover, it contained another embedded MZ with the same behavior, but designed for 64b architecture systems. The 64b launcher has nearly the same time stamp.

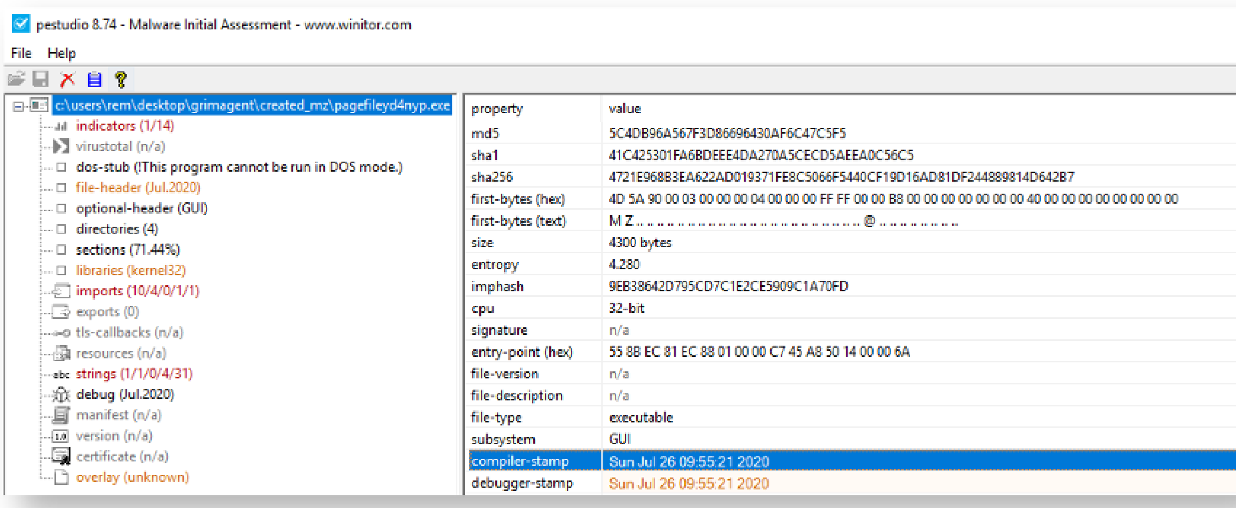


Fig. 8: MZ Shellcode/DLL launcher properties

The most interesting part of the new executable is how it executes the shellcode using sentinel bytes, with two possibilities:

1. @@@@@@@ + shellcode
2. @@@@@@d + DLL path + ':' + arguments (used at command 5)

In both cases, the malware parses the sentinel bytes and executes the shellcode/DLL.

The sentinel bytes are used to calculate where the shellcode is located and be able to execute the malicious code.

Once dropped, this MZ launcher will be executed through task scheduler with a random name of 8 alphanumeric characters.

By executing this command, the malware will also try to privesc by executing the payload as the highest privileges possible. It will then sleep for 195-205 seconds, after which the malware will delete the newly created task.

Command 3: Download and Execute (Schtask)

This is the basic functionality found in many malware families. It is the common download and execute command functionality.

In order to receive the payload after the request of a new command, GrimAgent performs a request to obtain the payload to the specified URI received as an argument. The payload will be dropped into a writable folder and executed through the task scheduler, as in command 1.

Command 4: Update

When this command is received, the malware updates itself by dropping the new updated GrimAgent file into its current directory and appending padding bytes (this action will change the file hash, used as a defensive mechanism) as well as the same *path_mutex_buff* at the end of the new binary. By appending these bytes, the malware creates the same mutex and checks for the same custom path for the config file. When finished, it runs the new updated binary (ShellExecuteW) and exits the process.

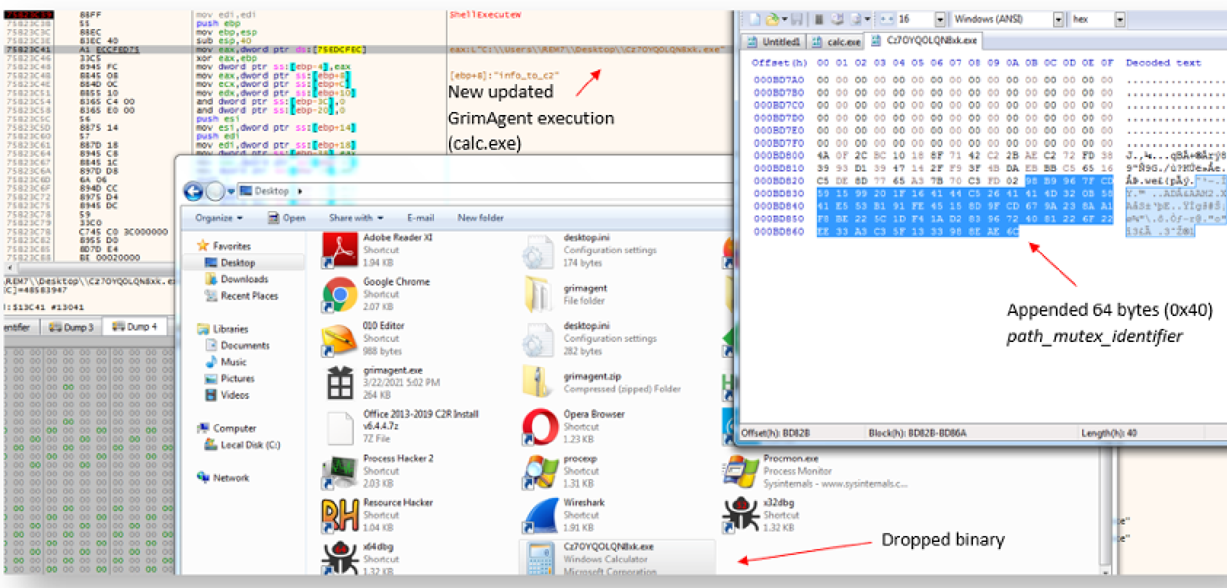


Fig. 9: Updated GrimAgent

Command 5: Execute DLL

(MZ launcher trampoline) This command is very similar to command 2. The function is called with three parameters:

1. Payload size to be received by C2 on the next request
2. DLL arguments

3. Bool if the DLL is 32b or 64b

As in previous commands, the malware will receive the URI to download the payload and, once executed, it will obtain a writable directory in order to be able to drop payloads. It will then perform a GET request to the URI in order to download the DLL to be executed.

Next, the malware will drop the DLL launcher (MZ) with the sentinel bytes "@@@@@@d" and append (at the next bytes) the path of the DLL to be executed + ":" + arguments. The file extension of the DLL launcher will be ".exe". On the other hand, it will also drop the DLL to be executed (with the file extension .dll).

Sentinel bytes syntax:

Once everything has been prepared, the malware will execute the launcher through the task scheduler and this will act as a stepping stone for executing the DLL, sending the previously obtained arguments.

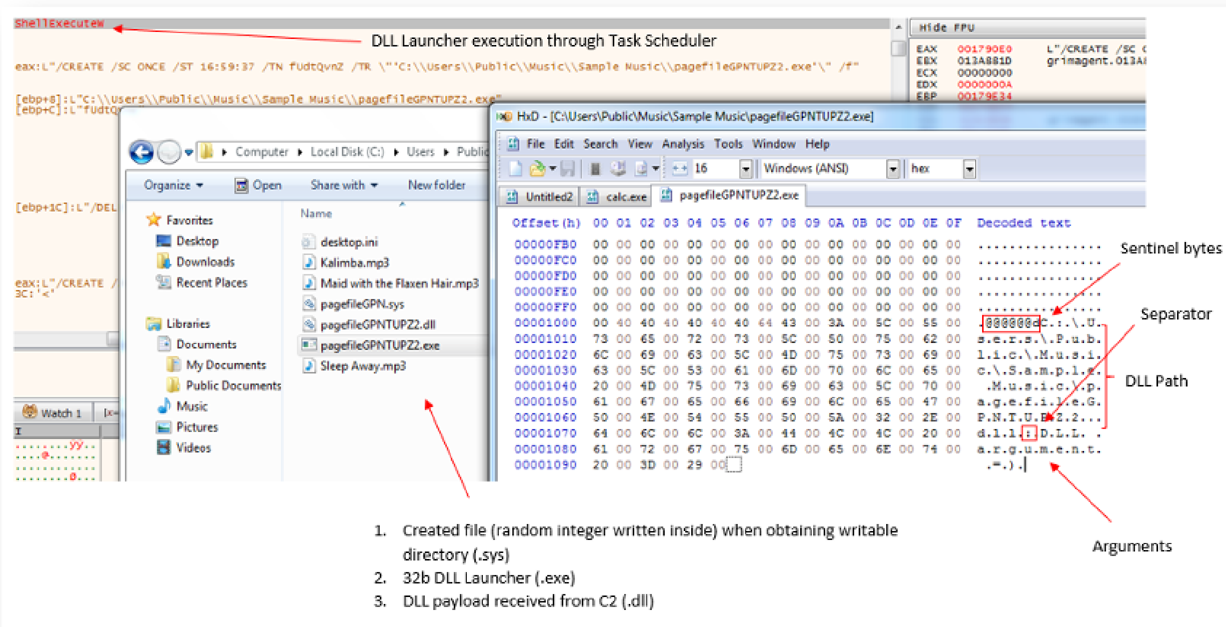


Fig. 10: DLL execution

Command 6: Download and Execute (ShellExecuteW)

This command obtains the payload by making a request to the URI (received as the command argument), drops it into the current directory and executes through a ShellExecuteW call.

This command was found in the newer versions of the malware.

Dealing with newer versions of GrimAgent

While working on this article, we found some **new GrimAgent samples in the wild with variations** from the version explained above.

As a sample of the different changes detailed below, we have the hash SHA256 - *D6EE553F52F20127301F737237B174EF6241EC9049AB22155DCE73144EF2354D,* which presents variations such as:

- **Hardcoded mutex name** "*T10*": This malware version does not generate a mutex name; instead, it creates one based on a hardcoded value.
- **Hardcoded config path** "*\Users\Public\microsoft.cfg*": As with the mutex, a path is no longer defined for the configuration; instead, it is hardcoded in the sample.
- **Copies itself into a hardcoded path and filename** (*"\Users\Public\svchost.exe"*), and then **sets persistence through the registry Run key**. The execution is done directly through the call *ShellExecuteW <path>* instead of schtask.
- **The function that searches writable directories** by creating and writing a new file with a random integer inside has been removed.
- **Hardcoded payload path**: Commands that used writable paths found now use a hardcoded path "*\Users\Public\system+<random>.exe*", for example command 3 (download and execute) and command 5 (execute DLL).
- **Added command 6** (drop and execute into the current path): The file will be executed through a ShellExecuteW call.
- **It does not contain *path_mutex_buffer*** (last 64 bytes in the binary).
- **Updated command 4**: The screenshot below compares the command 4 (update) of both GrimAgent versions. On the left is the old one (already explained in the article) and on the right is the updated one, which appends '00' bytes in the last 64 bytes of the updated binary and ignores whether or not it can read the last 64 bytes of the binary.

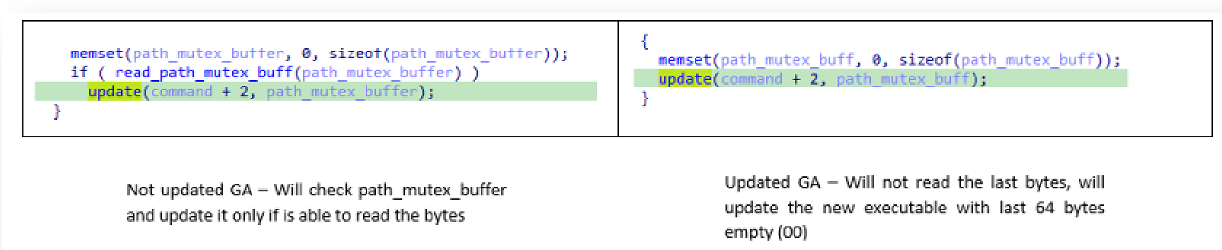


Fig. 11: Comparison of GrimAgent command 4 versions

The changes mean that **the malware is being actively developed** and it is highly likely that it will present even more different versions and changes over time. The fact that some forms of execution have been changed (such as the use of "*path_mutex_buffer*") does not mean that a threat actor will never use the former malware version again, so we must remain attentive and follow any new updates closely.

Network protocol

Base protocol

GrimAgents **implements a custom network protocol** in order to interact with the Command and Control server, and it follows the following syntax:

- MSG: The message to be sent.
- Flag: The value used to indicate what kind of connection needs to be made with the C2.
  - a: On requests for new commands. It makes a POST request to the Command and Control server.
  - d: First request to the C2. It makes a POST request to the Command and Control server to register the client on the server side.
  - **r:* It is used internally in the malware, although it is not added directly in the request to the C2. When the function responsible for contacting the Command and Control server is called with this flag, the malware will make a GET request to a URI in order to obtain (read) the payload (the next stage).
- Padding: Random generated values.
- Padding Size: The last two bytes are the decimal value of padding length - 3 (decimal format).

Randomization of connection fields

To establish a connection with the C2, GrimAgent randomizes some values as an evasive mechanism and tries to change request fields on the C2 request, such as User-Agent, referer, content-length and language.

As an interesting detail, if you look at the languages that the malware tries to adopt in the 'Accept-Language' field when connecting to the Command and Control server, it is possible that there is a relationship with the group's current targets. If so, they would be the following:

- United Kingdom
- United States
- Spain
- France

Detection opportunities

We have examined how GrimAgent behaves throughout its execution, when and what it does. We will now analyze the various opportunities to detect the malware. We can use the way in which it executes these actions to **monitor the behavior in the different defense mechanisms** or match them using Sigma, Yara or Suricata rules.

Detection opportunity 1: Persistence

On the first run, the malware copies itself to another directory, runs while establishing persistence on itself, and deletes the old file. A common path the malware uses is *C:/Users/Public*. GrimAgent carries out specific calls and they can be monitored to identify related behaviors.

Check the IOC section for the full information about the commands used on the malware persistence.

Detection opportunity 2: Mutex (old malware version)

One of GrimAgent's most characteristic factors is using the last 64 bytes of the binary to compute the name of the mutex. The characteristic can be used to create a behavior rule and therefore predict what mutex name it will create in the system.

All we have to do is to recreate the algorithm used in your different solutions by taking the last 64 bytes of the file and compute the possible mutex name. You can spot the algorithm used by the malware in the Mutex section

Detection opportunity 3: Network

The first domain to contact is *"api.myip.com*" in order to obtain the country code and client IP (*http://ip-api.com/csv/?fields=query,countryCode*) and then make the request to the C2 to obtain the AES key. Once finished, it will make periodic requests to the C2 infrastructure to obtain the following commands and/or to get next stage payloads. We can take advantage of the usage of the path "*/gate.php*" in conjunction with specific fields such as the referer (google.com, youtube.com, etc.) when contacting the C2.

Link to Suricata rule: https://github.com/apriegob/GrimAgent/blob/main/Ru...

Detection opportunity 4: Payload drop

While executing the malware commands related to executing shellcode and DLLs, it uses a binary embedded in the initial malware. We can create detection rules for this binary and alert our defense teams if a match is found.

The following Yara rule was created to detect shellcode and DLL launchers (32b/64b) embedded in GrimAgent.

Link to Yara rule: https://github.com/apriegob/GrimAgent/blob/main/Ru...

Detection opportunity 5: Payload execution

As we have shown throughout the article, to execute payloads GrimAgent uses both the ShellExecute call and indirect execution through scheduled tasks. Given that it always uses the same syntax on schtasks, we can try to match these actions. As it is suspicious behavior to create a scheduled task and try to execute it nearly at the same time as its creation, try to execute it with maximum privileges and delete it.

Check the IOC section for the full information about the commands used on the payload execution.

Hunt or be hunted

We have created various behaviors as well as static and network rules in order to be able to hunt and detect the GrimAgent malware family. The following screenshot shows Group-IB's Threat Hunting Framework (THF) and how the GrimAgent sample has been detonated and detected.

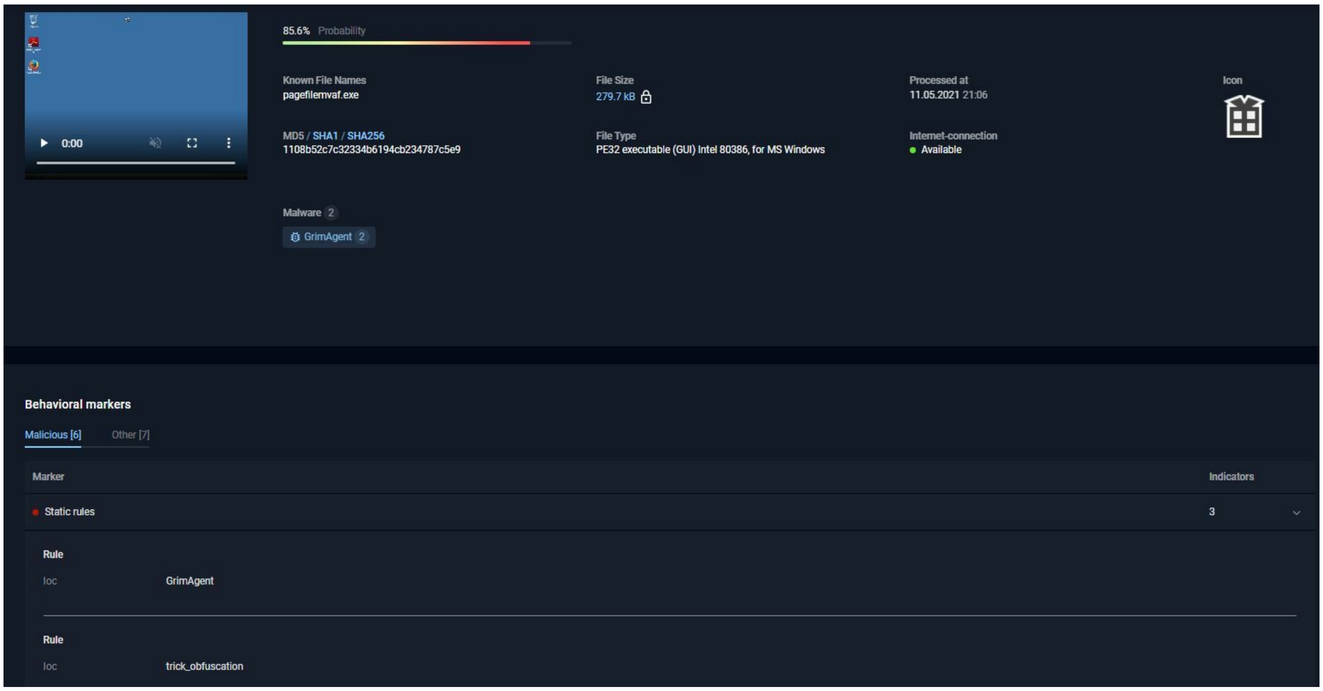


Fig. 12 - Group-IB THF detecting GrimAgent malware

Conclusion

There can be no doubt that we are facing a meticulous, careful and highly skilled adversary. Many have already fallen victim to Ryuk ransomware, which is known to target large companies, encrypt their content, and ask for large sums of money. When it comes to such threats, all defensive measures are not enough and we believe that we must direct our IR teams and companies in intelligence-based environments. We must get ahead of the adversary before they carry out their attack.

To help the cyber community better detect the adversary, IOC and Rules sections are provided below.

I would like to give a special thank you to my father, who has always supported me and lent me a helping hand when I have needed him. Thank you, always.

Albert

MITRE ATT&CK

GrimAgent's TTPs and relevant mitigation techniques in accordance with MITRE ATT&CK and MITRE Shield

GROUP|IB

Tactics	Technique	Sub-technique	Mitigations	Group-IB Solutions
Execution	Command and Scripting Interpreter - T1059	Windows Command Shell - T1059.003	Execution Prevention (M1038)	Threat Hunting Framework
	Native API - T1106		Execution Prevention (M1038)	
	Scheduled Task/Job - T1053	Scheduled Task - T1053.005	Audit (M1047), Operating System Configuration (M1028), Privileged Account Management (M1026), User Account Management (M1018)	
Persistence	Boot or Logon Autostart Execution- T1547	Registry Run Keys / Startup Folder - T1547.001		Threat Hunting Framework
	Scheduled Task/Job - T1053	Scheduled Task - T1053.005	Audit (M1047), Operating System Configuration (M1028), Privileged Account Management (M1026), User Account Management (M1018)	
Privilege Escalation	Scheduled Task/Job - T1053	Scheduled Task - T1053.005	Audit (M1047), Operating System Configuration (M1028), Privileged Account Management (M1026), User Account Management (M1018)	Threat Hunting Framework
Defense Evasion	Deobfuscate/Decode Files or Information - T1140			Threat Hunting Framework
	Indirect Command Execution - T1202			
	Masquerading - T1036	Masquerade Task or Service – T1036.004		
		Match Legitimate Name or Location – T1036.005	Code Signing (M1045), Execution Prevention (M1038), Restrict File and Directory Permissions (M1022)	
	Obfuscated Files or Information - T1027	Binary Padding - T1027.001		
		Software Packing - T1027.002	Antivirus/Antimalware (M1049)	
	Virtualization/Sandbox Evasion - T1497	Time Based Evasion - T1497.003		
Command and Control	Application Layer Protocol - T1071	Web Protocols - T1071.001	Network Intrusion Prevention (M1031)	Threat Hunting Framework
	Data Encoding - T1132	Standard Encoding - T1132.001	Network Intrusion Prevention (M1031)	
	Data Obfuscation - T1001	Junk Data - T1001.001	Network Intrusion Prevention (M1031)	
	Encrypted Channel - T1573	Symmetric Cryptography - T1573.001	Network Intrusion Prevention (M1031)	
		Asymmetric Cryptography - T1573.002	Network Intrusion Prevention (M1031), SSL/TLS Inspection (M1020)	
	Fallback Channels - T1008		Network Intrusion Prevention (M1031)	
	Ingress Tool Transfer - T1105		Network Intrusion Prevention (M1031)	
	Multi-Stage Channels - T1104		Network Intrusion Prevention (M1031)	
Collection	Archive Collected Data – T1560	Archive via Custom Method – T1560.003		Threat Hunting Framework
Exfiltration	Exfiltration Over C2 Channel - T1041		Network Intrusion Prevention (M1031)	Threat Hunting Framework
Impact	Data Encrypted for Impact - T1486		Data Backup (M1053)	Threat Hunting Framework

Group-IB, "The Brothers Grim: The reversing tale of GrimAgent malware used by Ryuk", 2021

Indicators of Compromise

Task scheduler

- /CREATE /SC ONSTART /TN [A-Za-z0-9]{4,9} /TR "<path to file>" /f
- /CREATE /SC ONSTART /TN [A-Za-z0-9]{4,9} /TR "<path to file>" /f /RL HIGHEST
- /CREATE /SC ONCE /ST hh:mm:ss /TN [A-Za-z0-9]{8} /TR "<path to file>" /f
- /CREATE /SC ONCE /ST hh:mm:ss /TN [A-Za-z0-9]{8} /TR "'<path to file> /f /RL HIGHEST

- /DELETE /TN [A-Za-z0-9]{8} /f

File deletion

cmd /c timeout 10 & del <path to file>

Registry

REG ADD HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /V "microsoft update" /t REG_SZ /F /D "SCHTASKS /run /tn [A-Za-z0-9]{8}"

Directories and files

- C:\Users\Public\config

- C:\Users\Public\reserve.sys

Command and Control server

- microsoftupdate[.]top/gate.php
- microsoftsystemcloud[.]com/gate.php
- chaseltd[.]top/gate.php

Hashes