Agrius Deploys Moneybird in Targeted Attacks Against Israeli Organizations

By etal

Published: 2023-05-24 · Archived: 2026-04-16 02:13:23 UTC

Key Points

- Iranian threat actor Agrius continues to operate against Israeli targets, masking destructive influence operations as ransomware attacks.
- In recent attacks the group deployed Moneybird, a previously unseen ransomware written in C++.
- Despite presenting themselves as a new group with the name– Moneybird, this is yet another Agrius alias.
- The data was eventually leaked through one of Agrius previous aliases.
- As demonstrated in the Moneybird attacks, Agrius's techniques, tactics and procedures (TTP) remain largely unchanged.

Introduction

While responding to a ransomware attack against an Israeli organization, the Check Point Incident Response Team (CPIRT) and CPR identified a new strain of ransomware called Moneybird. Although the payload itself was unique, the TTPs demonstrated in the attack had clear overlaps with a threat actor known as Agrius. The data was eventually leaked by an entity with one of the group's known aliases.

First introduced in 2021, Agrius is an Iran-aligned threat actor that operates mostly in the Middle-East. The actor has been tied to several ransomware and wiper attacks, with a major focus on Israeli institutions. The group's affiliation within Iran is not clear, although recent reports have tied it to the Iranian Ministry of Intelligence and Security (MOIS).

The newly discovered ransomware used by the group, Moneybird, was used to target organizations in Israel. This correlates with Agrius past activities against other organizations in Israel, most notably Shirbit and Bar Ilan University. The group has used a wide set of aliases for its extortion entities. BlackShadow, the name used by the group to extort Shirbit, was the first known alias Agrius has taken and is still commonly associated with it.

Agrius ransomware operations have been mostly tied to a custom ransomware called Apostle, which was originally a wiper. The use of a new ransomware, written in C++, is noteworthy, as it demonstrates the group's expanding capabilities and ongoing effort in developing new tools.

Activity Analysis

Agrius's actions leading to the deployment of Moneybird correlates to previous reports of the group's activity.


Figure 1 - High-level overview of Agrius activities leading to the deployment of Moneybird ransomware.
Figure 1 – High-level overview of Agrius activities leading to the deployment of Moneybird ransomware.

Agrius' first foothold was established by exploiting vulnerabilities within public-facing web servers, leading to the deployment of unique variants of ASPXSpy. The exploitation and the post-exploitation activities were carried out using public VPN services nodes, most prominently ProtonVPN nodes in Israel.

The ASPXSpy webshells were deployed in a unique fashion, hidden inside "Certificate" text files. This method is tied to past observed group activities. To use the webshell, the actor decoded the content of the file into a separate ASPX file.


Figure 2 - Webshell encoded within a fake certificate text file.
Figure 2 – Webshell encoded within a fake certificate text file.

Following the deployment of webshells, the threat actor was observed utilizing several publicly available tools to perform recon, move laterally, harvest credentials, and exfiltrate data. The tools include:

- **SoftPerfect Network Scanner** – Scan internal networks.
- **Plink** – RDP tunnel traffic from a VPS owned by the actor.
- **ProcDump** – Dump LSASS and harvest credentials. ****
- **FileZilla** – Exfiltrate compressed files.

Interestingly enough, the actor performed most of the activity while manually connected through RDP. To download some of the payloads, the actor opened a browser and connected to the legitimate file sharing services `ufile[.]io` and `easyupload[.]io` that hosted the malicious files.

One of the files the threat actor downloaded was the ransomware executable stored within an archive – Moneybird.

Moneybird Ransomware – Technical Analysis

Moneybird is written in C++ and contains an indicative PDB path: `C:\Users\user\Desktop\moneybird\x64\Release\moneybird.pdb`. The name embedded within the ransomware sample reveals that the encryptor shares the same name that appears in the attack ransom note for the attack: **Moneybird**.


Figure 3 - Moneybird ransom note.
Figure 3 – Moneybird ransom note.

Many recent ransomware strains typically support command-line parameters that enable attackers to customize malware functionality on top of the malware's embedded configuration. This specific threat lacks any command-line parsing capability. Instead, it includes a configuration blob embedded within the tool itself, which makes it less suitable for mass campaigns with different environments.


Figure 4 - Moneybird configuration.
Figure 4 – Moneybird configuration.

This configuration contains several key elements that are used when the malware is executed. The sample ignores the first DWORD. The second one contains an integer value representing the number of milliseconds the malware waits before executing.

After these initial values, the configuration includes four additional DWORDs.

- The first DWORD in this sequence acts as a flag, which determines whether the ransomware should take into account its embedded list of 194 extensions to target. The extension list includes the most common document formats, database formats, certificates, etc. A value of "**1**" means the list is consulted, while a value of "**2**" causes the ransomware to ignore the list and encrypt all files in the targeted paths indiscriminately, except for those kept in a narrow list of file extensions that are never encrypted: `exe, dll, sys, msi, lnk`.
- The second DWORD in the sequence contains the maximum number of threads that are created to encrypt every targeted file.
- The third DWORD contains the number of threads assigned to the routine that is executed before the encryption. This ensures that the targeted files are not marked as "**SYSTEM**" elements to be avoided.

> It is noteworthy to mention that the drawback of this approach is its lack of effectiveness in comparison to other ransomware strains that utilize the WIN API `GetSystemInfo` or directly access `PEB→dwNumberOfProcessors`. By using these methods, they can dynamically determine the number of CPUs per system and assign the encryption logic an appropriate number of threads based on this number.

- The final DWORD in the sequence specifies the size of an ASCII string that comes next. This string contains the base64-encoded public key that is used to encrypt the symmetric encryption keys that are generated per file.

Immediately following the public key, the configuration contains an integer value that determines the number of null-terminated strings that come next. These strings indicate the paths on the target machine that the sample encrypts, which is somewhat unusual as these malware usually try to cipher as much data as possible. In this particular case, there is only one path `F:\User Shares`, resulting in all other system paths being omitted. The remaining space in the configuration (up to 1024 bytes, including the previous elements) is reserved for possible additional system path entries. If no more entries are added, the remaining space is filled with the character "A", as in this sample.

If there is no path entry in the configuration, the malware behaves in a more generic fashion and uses the WIN API function `GetLogicalDrives` to obtain a list of currently available disk drives on the targeted machine and then starts to process it.

The configuration structure in decompiled C form:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

struct mb_config

{

DWORD start_sleep_delay;

DWORD ignore_extension_flag;

DWORD num_of_ciphering_threads;

DWORD num_of_check_threads;

DWORD sizeof_b64_public_key;

char b64_public_key[124];

DWORD num_off_paths;

char first_path[15];

char room_for_paths[860];

};

struct mb_config { DWORD start_sleep_delay; DWORD ignore_extension_flag; DWORD num_of_ciphering_threads; DWORD num_of_check_threads; DWORD sizeof_b64_public_key; char b64_public_key[124]; DWORD num_off_paths; char first_path[15]; char room_for_paths[860]; };

```
struct mb_config
{
        DWORD start_sleep_delay;
        DWORD ignore_extension_flag;
        DWORD num_of_ciphering_threads;
        DWORD num_of_check_threads;
        DWORD sizeof_b64_public_key;
        char b64_public_key[124];
        DWORD num_off_paths;
        char first_path[15];
        char room_for_paths[860];
};
```

The encryption logic of this ransomware sample depends on several embedded libraries, including “**libgcrypt**”, which is easily identifiable in the sample strings.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

...

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\mac.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\mpi\mpi-pow.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\src\fips.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\primegen.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\mpi\mpicoder.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\dsa.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\random\random-drbg.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\elgamal.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\blake2.c

C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\keccak.c

...

... C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\mac.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\mpi\mpi-pow.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\src\fips.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\primegen.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\mpi\mpicoder.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\dsa.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\random\random-drbg.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\elgamal.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\blake2.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\keccak.c ...

```
...
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\mac.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\mpi\mpi-pow.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\src\fips.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\primegen.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\mpi\mpicoder.c
```

```
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\dsa.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\random\random-drbg.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\elgamal.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\blake2.c
C:\Users\user\Desktop\moneybird\Shiftlibgcrypt\cipher\keccak.c
...
```

Looking at the folder name inside the strings, it is likely that the library was compiled from this GitHub repository. Basically, the repository contains an "unofficial" version of "**libgcrypt**", which the authors tried to make easier to include in Visual Studio projects.The malware also uses "**libpgp-error**", a library that libgcrypt requires as a dependency.

Finally, the malware is also linked with a copy of the "**cryptopp**" library. This library can be easily identified by strings that directly reference its name, as well as a distinctive test string that is used as text to encrypt in many versions of this library.


Figure 5 - cryptopp use in Moneybird.
Figure 5 – cryptopp use in Moneybird.

The ransomware uses the functions provided by the libraries to perform encryption using **AES-256** with **GCM** mode. The definition of both constants can be obtained from the source code of the library at the following link.


Figure 6 - Ransomware encryption using AES-256.
Figure 6 – Ransomware encryption using AES-256.

As you can see in the image above, the **IV** `012345678901255` for **AES-256-GCM** is hardcoded inside the ciphering function while the **key** is passed as the last parameter to the function.

The code responsible for the generation of the **key**:


Figure 7 - Key generation code.
Figure 7 – Key generation code.

This code is executed for each file, so each one is assigned a unique encryption key. To generate a key, the sample concatenates a **GUID** (marked in red) obtained through the WIN API `CoCreateGuid` with a **random number** (marked in green) generated using the `rand()` function. The seed for the `rand()` function is based on the system time. Then, **8 bytes of the file** content (marked in blue) to be encrypted is concatenated. Finally, the full **path for the target file** (marked in purple) is added, but only **4** bytes of it are used as the last part of the **key** as it completes the 32-byte chunk.


Figure 8 - Encryption key structure. As a result, this is what the key structure looks like:
Figure 8 – Encryption key structure.As a result, this is what the key structure looks like:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

struct aes_key

{

char guid[16];

int rand_val;

char file_content[8];

char file_path_start[4];

};

struct aes_key { char guid[16]; int rand_val; char file_content[8]; char file_path_start[4]; };

```
struct aes_key
{
        char guid[16];
      int rand_val;
      char file_content[8];
      char file_path_start[4];
};
```

The utilization of a **GUID** obtained through the WIN API `CoCreateGuid` makes it very difficult to obtain the encryption key, as it is generated by making an RPC call to "UuidCreate", which gets its randomness by calling `ProcessPrng` from `bcryptPrimitives.dll`, as this function is cryptographically secured to generate random bytes.

After the full path of the target file that overflows the 32-byte `aes_key`, the malware adds the length of the path, creating a kind of secondary structure:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

struct meta_info

{

char guid[16];

int rand_val;

char file_content[8];

char file_path[file_path_length];

int file_path_length;

};

struct meta_info { char guid[16]; int rand_val; char file_content[8]; char file_path[file_path_length]; int file_path_length; };

```
struct meta_info
{
        char guid[16];
      int rand_val;
      char file_content[8];
      char file_path[file_path_length];
        int file_path_length;
};
```

This structure is encrypted by the hybrid encryption system "**Elliptic Curve Integrated Encryption Scheme**" – CryptoPP ECIES – using the embedded public key shown previously inside the sample's configuration. After the file encryption, this encrypted `meta_info` structure is appended at the end of the final file, resulting in the struct below:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

struct encrypted_file

{

char enc_file_content[file_content_length];

char enc_meta_info[enc_meta_info_length];

int enc_meta_info_length;

};

struct encrypted_file { char enc_file_content[file_content_length]; char enc_meta_info[enc_meta_info_length]; int enc_meta_info_length; };

```
struct encrypted_file
{
      char enc_file_content[file_content_length];
      char enc_meta_info[enc_meta_info_length];
      int enc_meta_info_length;
};
```

Conclusion

Our analysis of incidents involving Moneybird reveals the ongoing effort of Agrius to utilize ransomware to make an impact. Although Agrius has used different aliases in the past, public reports up to now have tied most of their destructive activities to variants of Apostle, which acted as wipers or ransomware. The use of a new ransomware demonstrates the actor's additional efforts to enhance capabilities, as well as hardening attribution and detection efforts.

Despite these new "covers", the group continues to follow its usual behavior and utilize similar tools and techniques as before. Moneybird, like many other ransomware, is a grim reminder of the importance of good network hygiene, as significant parts of the activity could have been prevented early on.

Moneybird itself, although not particularly complex, has a number of intriguing features that appear to have been designed for specific targets. Some of these specialty features make the malware less practical for use in multiple unrelated campaigns. This emphasizes the malware's targeted nature, including the use of "targeted paths" which, in the specific sample we analyzed, makes the ransomware ignore most of the files on the target machine.

Check Point customers remain protected from the threats described in this research.

Check Point Threat Emulation provides comprehensive coverage of attack tactics, file types, and operating systems, and has developed and deployed a signature named "Ransomware.Wins.MoneyBird" to detect and protect our customers against Moneybird.

YARA

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

rule ransomware_moneybird {

meta:

author = "Marc Salinas @ Check Point Research"

description = "Detects a ransomware sample named Moneybird based on its pdb string."

malware_family = "MoneyBird"

date = "11/05/2023"

sample = "aa19839b1b6a846a847c5f4f2a2e8e634caeebeeff7af59865aecca1d7d9f43c"

strings:

$ran1 = "WE ARE MONEYBIRD!"

$ran2 = "All of your data encrypted!"

$ran3 = "ok.ru/profile"

$ext1 = "Shiftlibgcrypt"

$ext2 = "come to the aide of their"

$ext3 = "stopmarker" wide

$code1 = {44 89 4C 24 20 4C 89 44 24 18 48 89 54 24 10 89 4C 24 08 56 57 48 83 EC 78 48 8B 05 68 FE 1A 00 48 33 C4 48 89 44 24 60 48 8D 44 24 50 48 8D 0D DC 68 15 00 48 8B F8 48 8B F1 B9 10 00 00 00 F3 A4 45 33 C9 41 B8 09 00 00 00 BA 09 00 00 00 48 8D 4C 24 48 ?? ?? ?? ?? ?? 41 B8 20 00 00 00 48 8B 94 24 A0 00 00 00 48 8B 4C 24 48 ?? ?? ?? ?? ?? 48 8D 44 24 50 48 89 44 24 40 48 C7 44 24 30 FF FF FF FF}

$code2 = {48 FF 44 24 30 48 8B 44 24 40 48 8B 4C 24 30 80 3C 08 00}

$code3 = {48 8B 44 24 30 4C 8B C0 48 8D 54 24 50 48 8B 4C 24 48 ?? ?? ?? ?? ?? 8B 84 24 90 00 00 00 48 C7 44 24 20 00 00 00 00 45 33 C9 44 8B C0 48 8B 94 24 98 00 00 00 48 8B 4C 24 48 ?? ?? ?? ?? ?? 89 44 24 38 48 8B 4C 24 48 ?? ?? ?? ?? ?? 48 8B 4C 24 60 48 33 CC ?? ?? ?? ?? ?? 48 83 C4 78 5F 5E C3}

condition:

uint16(0) == 0x5A4D and (2 of ($ran*) or all of ($code*) or all of ($ext*))

rule ransomware_moneybird { meta: author = "Marc Salinas @ Check Point Research" description = "Detects a ransomware sample named Moneybird based on its pdb string." malware_family = "MoneyBird" date = "11/05/2023" sample = "aa19839b1b6a846a847c5f4f2a2e8e634caeebeeff7af59865aecca1d7d9f43c" strings: $ran1 = "WE ARE MONEYBIRD!" $ran2 = "All of your data encrypted!" $ran3 = "ok.ru/profile" $ext1 = "Shiftlibgcrypt" $ext2 = "come to the aide of their" $ext3 = "stopmarker" wide $code1 = {44 89 4C 24 20 4C 89 44 24 18 48 89 54 24 10 89 4C 24 08 56 57 48 83 EC 78 48 8B 05 68 FE 1A 00 48 33 C4 48 89 44 24 60 48 8D 44 24 50 48 8D 0D DC 68 15 00 48 8B F8 48 8B F1 B9 10 00 00 00 F3 A4 45 33 C9 41 B8 09 00 00 00 BA 09 00 00 00 48 8D 4C 24 48 ?? ?? ?? ?? ?? 41 B8 20 00 00 00 48 8B 94 24 A0 00 00 00 48 8B 4C 24 48 ?? ?? ?? ?? ?? 48 8D 44 24 50 48 89 44 24 40 48 C7 44 24 30 FF FF FF FF} $code2 = {48 FF 44 24 30 48 8B 44 24 40 48 8B 4C 24 30 80 3C 08 00} $code3 = {48 8B 44 24 30 4C 8B C0 48 8D 54 24 50 48 8B 4C 24 48 ?? ?? ?? ?? ?? 8B 84 24 90 00 00 00 48 C7 44 24 20 00 00 00 00 45 33 C9 44 8B C0 48 8B 94 24 98 00 00 00 48 8B 4C 24 48 ?? ?? ?? ?? ?? 89 44 24 38 48 8B 4C 24 48 ?? ?? ?? ?? ?? 48 8B 4C 24 60 48 33 CC ?? ?? ?? ?? ?? 48 83 C4 78 5F 5E C3} condition: uint16(0) == 0x5A4D and (2 of ($ran*) or all of ($code*) or all of ($ext*))

```
rule ransomware_moneybird {
  meta:
    author = "Marc Salinas @ Check Point Research"
    description = "Detects a ransomware sample named Moneybird based on its pdb string."
    malware_family = "MoneyBird"
    date = "11/05/2023"
    sample = "aa19839b1b6a846a847c5f4f2a2e8e634caeebeeff7af59865aecca1d7d9f43c"

  strings:
    $ran1 = "WE ARE MONEYBIRD!"
    $ran2 = "All of your data encrypted!"
    $ran3 = "ok.ru/profile"

    $ext1 = "Shiftlibgcrypt"
    $ext2 = "come to the aide of their"
    $ext3 = "stopmarker" wide

    $code1 = {44 89 4C 24 20 4C 89 44 24 18 48 89 54 24 10 89 4C 24 08 56 57 48 83 EC 78 48 8B 05 68
    $code2 = {48 FF 44 24 30 48 8B 44 24 40 48 8B 4C 24 30 80 3C 08 00}
    $code3 = {48 8B 44 24 30 4C 8B C0 48 8D 54 24 50 48 8B 4C 24 48 ?? ?? ?? ?? ?? 8B 84 24 90 00 00

  condition:
    uint16(0) == 0x5A4D and (2 of ($ran*) or all of ($code*) or all of ($ext*))
```

Source: https://research.checkpoint.com/2023/agrius-deploys-moneybird-in-targeted-attacks-against-israeli-organizations/