A detailed analysis of the Money Message Ransomware

Prepared by: Vlad Pasca, Senior Malware & Threat Analyst
# Table of contents

- Executive summary .................................................. 2
- Analysis and findings .............................................. 2
- Thread activity – sub_ED2770 function ......................... 9
- Thread activity – sub_F179D0 function ......................... 10
- Thread activity – sub_FIDICO function ......................... 12
- Running with the --crypt parameter ........................... 17
- Running with the -d parameter ................................... 17
- Running with the -l parameter ................................... 18
- Running with the -v parameter ................................... 18
- Indicators of Compromise .......................................... 19
Executive summary

The threat actor group, Money Message ransomware, first appeared in March 2023, demanding million-dollar ransoms from its targets. Its configuration, which contains the services and processes to stop a ransomware attack, can be found at the end of the executable. The ransomware creates a mutex and deletes the Volume Shadow Copies using `vssadmin.exe`.

The files are encrypted using the ChaCha20 algorithm, with the key being encrypted using ECDH (Elliptic-curve Diffie-Hellman). The extension of the encrypted files isn’t changed, however the structure of the files indicates they were encrypted.

Analysis and findings

SHA256: 8be41ef666e6ace53b8c59344be8ba9fe41003987a8e38484b20760d7c400a42

The malware decrypts a list of arguments that it can run with: "--crypt", "-d", "-l", and "-v". We’ll explain the purpose of each argument in our analysis.

![Figure 1](image-url)

The messages that would be displayed in the console are decrypted using the same SUB instruction (see Figure 2).
The ransomware retrieves the current system date and time via a function call to `GetSystemTimePreciseAsFileTime`:

The `GetCurrentThreadId` API is utilized to obtain the ID of the calling thread:

The process writes relevant strings in the console by calling the `WriteConsoleA` method:
It extracts information about the console screen buffer using GetConsoleScreenBufferInfo:

The malware changes the text color for output using the SetConsoleTextAttribute function (0x2 = FOREGROUND_GREEN):

The executable file is opened by calling the wfsopen method (0x40 = _SH_DENYNO):

The binary is looking for its configuration by reading 4096 bytes at a time:

The configuration contains the ransom note content, the mutex name, a list of directories that will be skipped, the public key, a list of processes and services to stop, a list of corporate credentials previously extracted from the victim, and the temporary extension:
The directories extracted from the “skip_directories” field are Base64-decoded (Figure 12).
The ransomware creates a mutex called “12345-12345-12235-12354”, which ensures that only one copy is running at a single time:

Figure 13

It opens the “ServicesActive” database using the OpenSCManagerW API (0x80000000 = GENERIC_READ):

Figure 14

The malicious binary obtains a list of all services that run in their own processes using EnumServicesStatusExW (0x10 = SERVICE_WIN32_OWN_PROCESS, 0x3 = SERVICE_STATE_ALL):

Figure 15

The following services will be stopped:

- "vs" "sql" "svc$" "memtas" "mepocs" "sophos" "veeam" "backup" "vmms"
The malware takes a snapshot of all processes via a call to CreateToolhelp32Snapshot (0x2 = \texttt{TH32CS_SNAPPROCESS}): 

![Figure 17](image)

The processes are enumerated using the Process32FirstW and Process32NextW APIs, as shown below:

![Figure 18](image)

![Figure 19](image)

The ransomware opens the target processes using the OpenProcess method (0x1 = \texttt{PROCESS_TERMINATE}): 

![Figure 20](image)

The following processes will be killed:

- "sql.exe" "oracle.exe" "ocssd.exe" "dbsnmp.exe" "syntime.exe" "agntsvc.exe" "isqlplussvc.exe" "xfsvccon.exe" "mydesktopservice.exe" "oautoupds.exe" "encsvc.exe" "firefox.exe" "tbirdconfig.exe" "mdesktopqos.exe" "ocomm.exe" "dbeng50.exe" "sqbcoreservice.exe" "excel.exe" "infopath.exe" "msaccess.exe" "mspub.exe" "onenote.exe" "outlook.exe" "powerpnt.exe" "steam.exe" "thebat.exe" "thunderbird.exe" "visio.exe" "winword.exe" "wordpad.exe" "vmms.exe" "vmwp.exe"

![Figure 21](image)
The following directories will not be encrypted:

- "C:\msocache"
- "C:\$windows.~ws"
- "C:\system volume information"
- "C:\perflogs"
- "C:\programdata"
- "C:\program files (x86)"
- "C:\program files"
- "C:\$windows.~bt"
- "C:\windows"
- "C:\windows.old"
- "C:\boot"

The executable retrieves a pseudo handle for the process, as highlighted in Figure 22.

![Figure 22](image)

IsWow64Process is used to determine if the process is running on a 64-bit architecture:

![Figure 23](image)

The ransomware disables file system redirection for the current thread (see Figure 24).

![Figure 24](image)

It deletes all Volume Shadow Copies using the vssadmin.exe tool:

![Figure 25](image)

A new thread is created, which runs the sub_ED2770 function even if the argument passed to CreateThread is the StartAddress function:

![Figure 26](image)

![Figure 27](image)
Thread activity – sub_ED2770 function

The LookupPrivilegeValueA API is utilized to obtain the LUID for the following privileges: "SeAssignPrimaryTokenPrivilege", "SeRestorePrivilege", and "SeTakeOwnershipPrivilege":

![Figure 28](image1)

The executable opens the access token associated with the current process (0xF01FF = \texttt{TOKEN\_ALL\_ACCESS}):

![Figure 29](image2)

The process enables all the privileges mentioned above by calling the AdjustTokenPrivileges method, as highlighted below:

![Figure 30](image3)

The ransomware retrieves a list of sessions on the machine using \texttt{WTSEnumerateSessionsW} (Figure 31).

![Figure 31](image4)

For each of the identified sessions, the malware obtains the access token of the logged-on user:

![Figure 32](image5)
The DuplicateTokenEx method is used to create an impersonation token that duplicates the above token (0x2 = **SecurityImpersonation**, 0x2 = **TokenImpersonation**):

![Figure 33](image)

The malicious binary creates a new thread that will identify the shared resources. The credentials extracted from the configuration will be used to access those shares.

![Figure 34](image)

**Thread activity – sub_F179D0 function**

The executable extracts a pseudo handle for the current thread:

![Figure 35](image)

SetThreadToken is utilized to assign the impersonation token to the current thread, as displayed in Figure 36.

![Figure 36](image)

The ransomware starts enumerating all currently connected resources via a function call to WNetOpenEnumW (0x1 = **RESOURCE_CONNECTED**):

![Figure 37](image)

The enumeration continues using the WNetEnumResourceW API (see Figure 38). The malware is looking for files to encrypt in these network resources.
We continue to analyze the main thread.

The process iterates over drives in the range “Z:” to “A:” and calls the GetDriveTypeW method:

For each of the identified drives, the ransomware calls the CreateFileW function (0x80 = FILE_READ_ATTRIBUTES, 0x7 = FILE_SHARE_DELETE | FILE_SHARE_WRITE | FILE_SHARE_READ, 0x3 = OPEN_EXISTING, 0x02000000 = FILE_FLAG_BACKUP_SEMANTICS):

The binary obtains the final path for the drives by calling the GetFinalPathNameByHandleW API:

It extracts information about the current system using GetNativeSystemInfo (see Figure 42).

The ransomware creates the ransom note called “money_message.log” in every drive. The file contains the chat ID that is specific to a victim and can be used to contact the threat actor:
The malware creates a new thread that handles the files encryption:

**Thread activity – sub_F1D1C0 function**

The ransomware opens the directory to encrypt using CreateFileW, as shown in the figure below.

The files are enumerated using the FindFirstFileExW and FindNextFileW functions:
The following files will not be encrypted:

- "desktop.ini"  
- "ntuser.dat"  
- "thumbs.db"  
- "iconcache.db"  
- "ntuser.ini"  
- "ntldr"  
- "bootfont.bin"  
- "ntuser.dat.log"  
- "bootsect.bak"  
- "boot.ini"  
- "autorun.inf"

The malware retrieves attributes for a file to be encrypted by calling the GetFileAttributesExW method:

```
Figure 49
```

The ECDH public key used is hard-coded in the executable "7182b09d7f950841835ed0fc926a7f6888be7af4fc07ddc06dd8a1e6400bf3ad4cc27083aad
e393c52625708cbf47aa9855fbb9ecc0c82be053c8d957f74af2e32d8005e7059461b1e958fecd7dcb1fa36bf9f9bc8746d431902fe990772d16babb779d6e626544401001708d091df3838c0a15b0ccf9
9db70e951a74670dd0eb719678d62d150e0ee2706".

CryptAcquireContextA is utilized to acquire a handle to a key container within a cryptographic service provider (0x1 = **PROV_RSA_FULL**, 0xF0000040 = **CRYPT_VERIFYCONTEXT** | **CRYPT_SILENT**):

```
Figure 50
```

**CSPRNG** is used to generate 0x48 random bytes. These bytes, together with the ECDH public key, will be used to generate the shared secret.

```
Figure 51
```
The shared secret generated between the public key and the random bytes is 144 bytes long. The elliptic curve is $E_384$ for the ECDH algorithm.
The SHA384 algorithm implementation is shown in Figure 56. The process computes the hash of the shared secret and copies the first 32 resulting bytes to a new buffer. These bytes represent the ChaCha20 key that will be used to encrypt the file. The nonce (16 bytes) is randomly generated using the same library.

The binary creates an intermediary file by adding the “cbgnfvn” string at the end of the filename (0xC0000000 = GENERIC_READ | GENERIC_WRITE, 0x3 = FILE_SHARE_READ | FILE_SHARE_WRITE, 0x3 = OPEN_EXISTING, 0x80 = FILE_ATTRIBUTE_NORMAL):

The GetFileType method is utilized to obtain the file type:
The ransomware moves the file pointer to the beginning of the file (0x0 = FILE_BEGIN):

![Figure 59](image)

The file content is read via a function call to ReadFile (see Figure 60).

![Figure 60](image)

The content is encrypted using the ChaCha20 algorithm:

![Figure 61](image)
The encrypted file content is written back to the file using WriteFile (Figure 63).

![Figure 63](image)

The encrypted files extension is changed back to the original after the encryption is complete. The operation is done using MoveFileExW (0x3 = MOVEFILE_COPY_ALLOWED | MOVEFILE_REPLACE_EXISTING):

![Figure 64](image)

The ChaCha20 key is encrypted using ECDH and written to the encrypted file. The ChaCha20 nonce is stored in a non-encrypted form:

![Figure 65](image)

**Running with the --crypt parameter**

The malware only encrypts the directory passed as the argument.

**Running with the -d parameter**

In this case, the ransomware doesn’t stop the target services and processes and doesn’t delete the Volume Shadow Copies.
Running with the `-l` parameter

The process creates a log file called “encrypt_log.txt” that stores the messages written to the console.

Running with the `-v` parameter

This is the verbose mode that displays all the intermediary steps during the malware’s execution:

```
Figure 66
```
Indicators of Compromise

SHA256
8be41efd6e6ace53b8c59344be2ba91fe41003987a8e38484b20760d7c400a42

Money Message Ransom Note
money_message.log

Mutex
12345-12345-12235-12354

Process spawned

```
cmd.exe /c vssadmin.exe delete shadows /all /quiet
```