Breaking the B0 ransomware: Investigation & Decryption

porthas.com/blog/b0-ransomware-decryption/

April 17, 2025

Breaking the unbreakable: Story of B0 software group ransomware

This is how deep technical curiosity, DFIR expertise, and persistence led to the downfall of B0 Software Group's ransomware. In this article, we walk through the full investigation from start to finish: from the initial response and behavioral analysis to reverse engineering and decryptor development. With annotated code snippets, screenshots, and step-by-step breakdowns, we reveal not just what was done, but why it worked.

If you're looking for a real-world look at how threat intelligence and malware analysis can turn the tide in ransomware recovery, this is it.

Known Tactics, Techniques, and Procedures

Since ransomware groups can often sell their malware to multiple affiliates. These affiliates can have different flavors and attack methodologies for how they breach a targeted network. As such, there is no single set of TTPs that can generally cover how B0 software group ransomware can end up on your network. In this case, however, the DFIR investigation highly suggests that brute-forcing an internet-facing RDP (RDP brute-forcing) was the initial entry point.

The assessment was based on the following factors:

- Evidence of a brute force attack. Found repeated failed logins to the Administrator account on the compromised server. The brute force attempts originated from external IP addresses.

- The default RDP port 3389 was directly exposed to the internet at the time of the incident. The default RDP port is frequently an attractive attack vector and can be easily discovered by automated port scanners such as Nmap.

DFIR Investigation Conclusion

As for activities within the network and lateral movement, the DFIR investigation concluded the following points:

- Network discovery/reconnaissance. Upon gaining access to the server, the threat actor used the network reconnaissance tools Advance IP Scanner and kportscan 3.0 to identify additional devices to which to move laterally. However, because the server was the only device within a segmented, isolated environment, further lateral movement was unsuccessful.
- Account usage and privilege escalation. The threat actor gained access to the privileged Administrator account on the server via a brute force attack. Upon successfully logging into the system, they also created the account B0Us. The threat actor also executed the credential harvesting tool, Mimikatz, on the server. This is a credential dumping tool, typically used by threat actors to obtain any available credentials stored within the memory of a target system.
- Ransomware deployment. The threat actor transferred the compressed file 'pth.rar' to the system, which contained their toolkit, including the ransomware executable 'b0.exe'. The compressed file was initially placed in the directory 'C:\users\b0us\desktop\' and then subsequently extracted using the file compression tool, WinRAR. The threat actor placed their ransomware binary within the folder 'C:\users\public', manually executing this on the server leading to the encryption of files on the system and creation of ransom notes.

Exfiltration Analysis

We assess that the threat actor **did not exfiltrate data** from the impacted server. This assessment is based on the following factors:

- No evidence of file transfer tooling. S-RM found no evidence of data transfer tooling that could have been used by the threat actor to exfiltrate data from the server.
- No evidence of data access or staging activities. S-RM reviewed data extracted from the system and did not identify any conclusive evidence typically preceding exfiltration, such as data reconnaissance and compression. While we found evidence of interaction with the server's data drives, it is likely that this occurred in the context of pre-encryption activities.
- Threat actor claims. The threat actor did not claim to have exfiltrated any data from the server in their ransom note. Further, S-RM found no evidence of a leak site maintained by the threat actor to publish victim data as part of data extortion tactics.

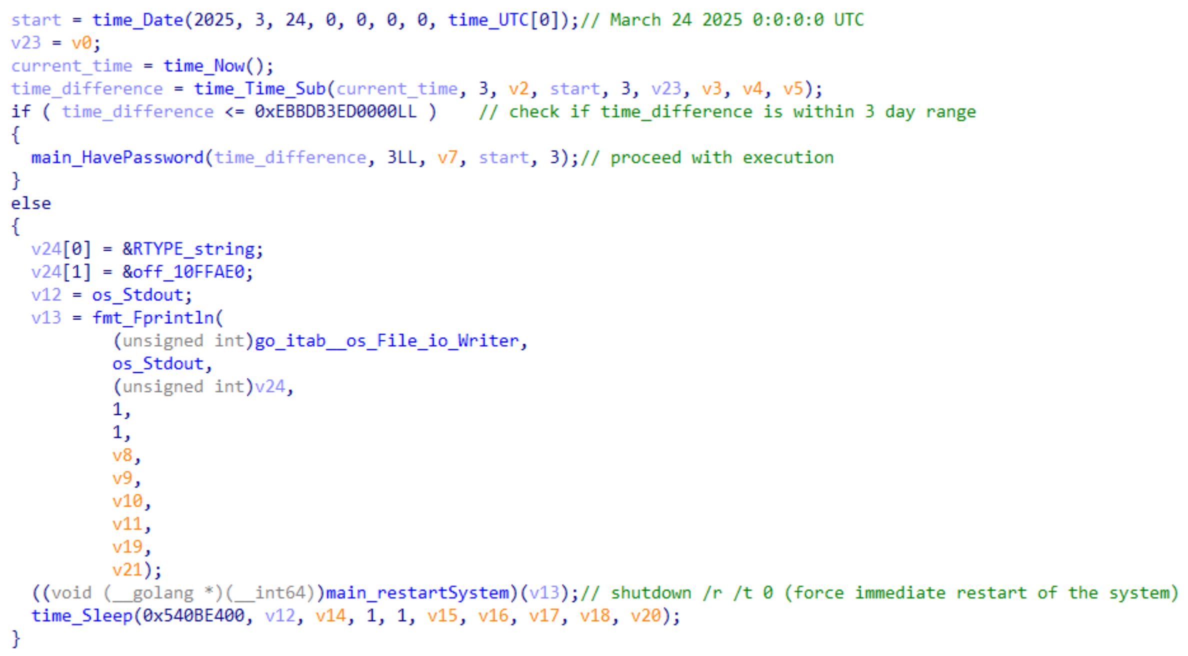
Limitations

- Due to the hosted nature of the server, S-RM could not obtain access to network or firewall logs, typically providing insight into data traffic and volumes. We were therefore unable to determine whether there any significant spikes in outbound data during the timeframe of the incident consistent with exfiltration activities.
- The evidence indicates that a limited number of files were accessed by the threat actor, however there is no accompanying evidence that would show that this data was transferred out of the system.

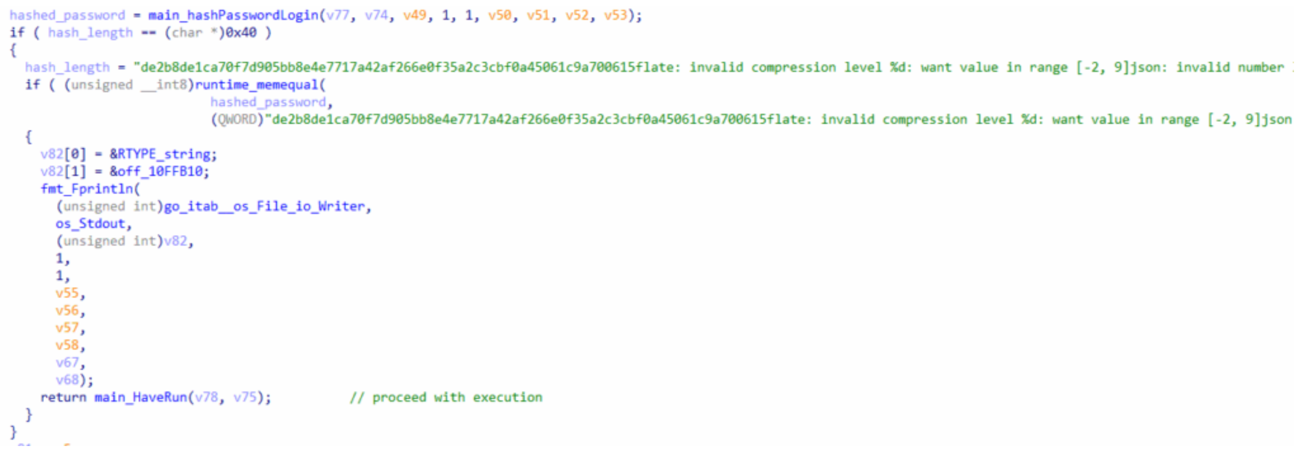
Technical analysis

The ransomware was written in Golang, making it more challenging to reverse engineer compared to other ransomware variants which are in C/C++.

The compilation timestamp of the analyzed sample was set to zero, which might have been a tactic used by the authors to confuse analysts and make it more difficult to determine when the sample was compiled. Notably, the **ransomware includes its own kill switch**. It calculates the time difference between a hardcoded UTC timestamp of **March 24, 2025, at 00:00:00**, and the time when the sample is executed. The ransomware checks if this time difference falls within a 3-day range. If it does, the ransomware continues its normal execution. If not, it immediately restarts the system by executing the command **“shutdown /r /t 0”**.

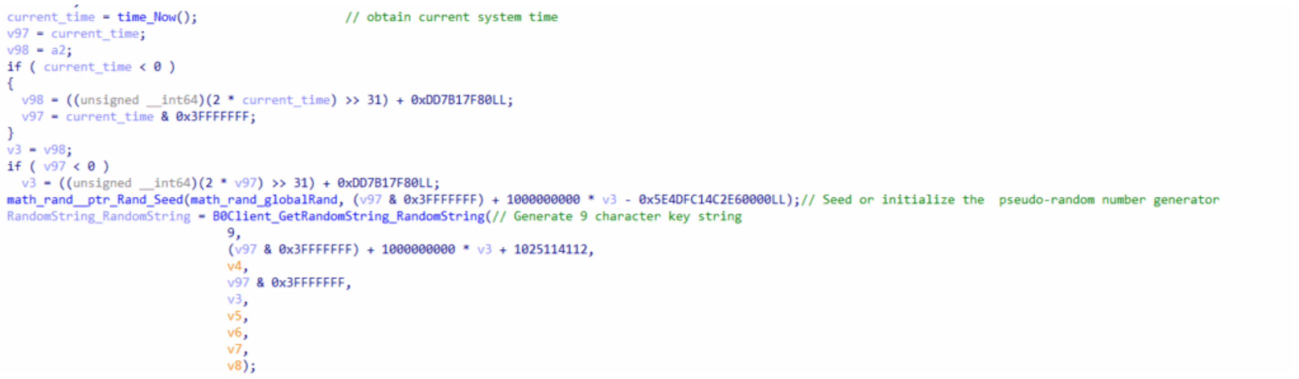

```
start = time_Date(2025, 3, 24, 0, 0, 0, 0, time_UTC[0]);// March 24 2025 0:0:0:0 UTC
v23 = v0;
current_time = time_Now();
time_difference = time_Time_Sub(current_time, 3, v2, start, 3, v23, v3, v4, v5);
if ( time_difference <= 0xEBBDB3ED0000LL )    // check if time_difference is within 3 day range
{
  main_HavePassword(time_difference, 3LL, v7, start, 3);// proceed with execution
}
else
{
  v24[0] = &RTYPE_string;
  v24[1] = &off_10FFAE0;
  v12 = os_Stdout;
  v13 = fmt_Fprintln(
          (unsigned int)go_itab__os_File_io_Writer,
          os_Stdout,
          (unsigned int)v24,
          1,
          1,
          v8,
          v9,
          v10,
          v11,
          v19,
          v21);
  ((void (__golang *)(__int64))main_restartSystem)(v13);// shutdown /r /t 0 (force immediate restart of the system)
  time_Sleep(0x540BE400, v12, v14, 1, 1, v15, v16, v17, v18, v20);
}
```

After confirming that the process runs within the allowed time range, it moves on to another check. During this check, it prompts the user for a password and a network ID. The provided password is then hashed using SHA-256 and compared against a hardcoded hash. If the hashes match, the execution will proceed; otherwise, it will prompt for the password again, up to three times. If all three attempts fail, the process will exit.



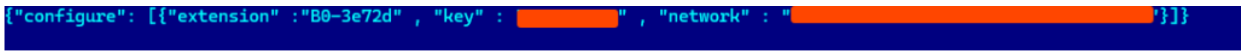
```
hashed_password = main_hashPasswordLogin(v77, v74, v49, 1, 1, v50, v51, v52, v53);
if ( hash_length == (char *)0x40 )
{
  hash_length = "de2b8de1ca70f7d905bb8e4e7717a42af266e0f35a2c3cbf0a45061c9a700615flate: invalid compression level %d: want value in range [-2, 9]json: invalid number
  if ( (unsigned __int8)runtime_memequal(
                          hashed_password,
                          (QWORD)"de2b8de1ca70f7d905bb8e4e7717a42af266e0f35a2c3cbf0a45061c9a700615flate: invalid compression level %d: want value in range [-2, 9]json
  {
    v82[0] = &RTYPE_string;
    v82[1] = &off_10FFB10;
    fmt_Fprintln(
      (unsigned int)go_itab__os_File_io_Writer,
      os_Stdout,
      (unsigned int)v82,
      1,
      1,
      v55,
      v56,
      v57,
      v58,
      v67,
      v68);
    return main_HaveRun(v78, v75);          // proceed with execution
  }
}
```

Upon successfully completing all the previously mentioned checks, the process moves on to configure the setup. First, it initializes a pseudo-random number generator using the current system time obtained by calling "time.Now.UnixNano()." Next, it generates a random nine-character string that will later be used for the key derivation function. This key will be crucial for the decryption and recovery of ransomware, which we will discuss later.



```
current_time = time_Now();                    // obtain current system time
v97 = current_time;
v98 = a2;
if ( current_time < 0 )
{
  v98 = ((unsigned __int64)(2 * current_time) >> 31) + 0xDD7B17F80LL;
  v97 = current_time & 0x3FFFFFFF;
}
v3 = v98;
if ( v97 < 0 )
  v3 = ((unsigned __int64)(2 * v97) >> 31) + 0xDD7B17F80LL;
math_rand__ptr_Rand_Seed(math_rand_globalRand, (v97 & 0x3FFFFFFF) + 1000000000 * v3 - 0x5E4DFC14C2E60000LL);// Seed or initialize the  pseudo-random number generator
RandomString_RandomString = B0Client_GetRandomString_RandomString(// Generate 9 character key string
                              9,
                              (v97 & 0x3FFFFFFF) + 1000000000 * v3 + 1025114112,
                              v4,
                              v97 & 0x3FFFFFFF,
                              v3,
                              v5,
                              v6,
                              v7,
                              v8);
```

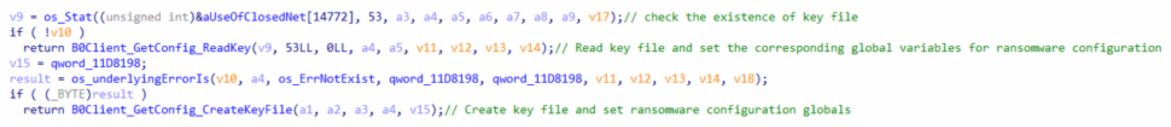
Next, it will verify the existence of the JSON file "telemetry.ASM-WindowsDefault.json" located in "C:\Windows\System32". This JSON file contains the ransomware configuration, including the initial key (a random nine-character string), the extension, and the Network ID.



```
{"configure": [{"extension" :"B0-3e72d" , "key" : [redacted]" , "network" : "[redacted]"}]}
```

JSON config file containing the extension, key and the network ID.

If the file is found, it continues by reading the file and setting the corresponding global variables for ransomware configuration.

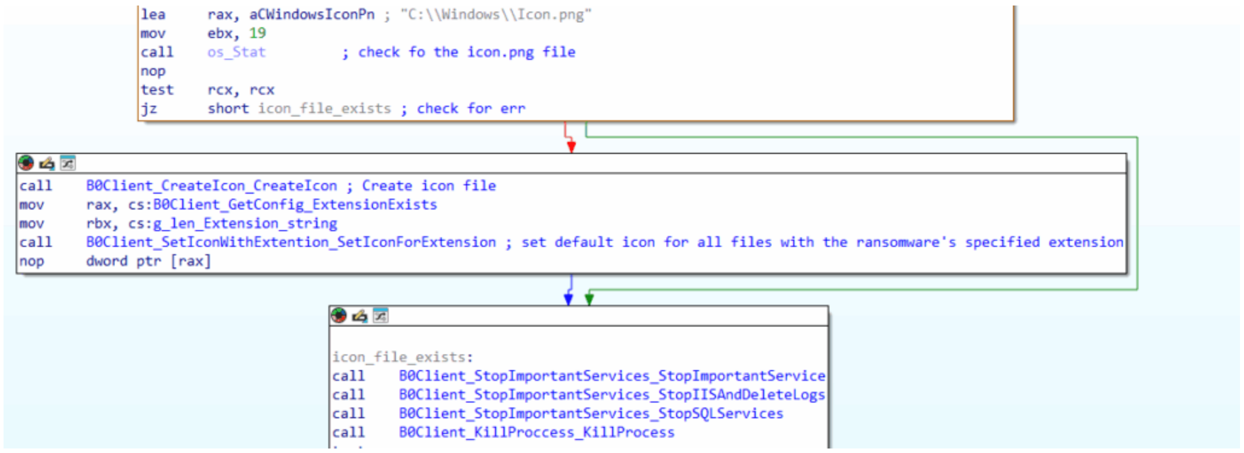


```
v9 = os_Stat((unsigned int)&aUseOfClosedNet[14772], 53, a3, a4, a5, a6, a7, a8, a9, v17);// check the existence of key file
if ( !v10 )
  return B0Client_GetConfig_ReadKey(v9, 53LL, 0LL, a4, a5, v11, v12, v13, v14);// Read key file and set the corresponding global variables for ransomware configuration
v15 = qword_11D8198;
result = os_underlyingErrorIs(v10, a4, os_ErrNotExist, qword_11D8198, qword_11D8198, v11, v12, v13, v14, v18);
if ( (_BYTE)result )
  return B0Client_GetConfig_CreateKeyFile(a1, a2, a3, a4, v15);// Create key file and set ransomware configuration globals
```

Configuration check and setup.

Similar to the key file, it will check for the existence of a file “Icon.png”, located in the Windows directory “C:\Windows”. Same as before, if the file is not found, it proceeds to create it. The icon file contents are stored as base64 encoded text that will be decoded and written to the file at run-time.

If the file was not found, it then creates the file “Icon.png” and sets the default icon for files with the ransomware’s extension (as specified in the ransomware configuration) to this “Icon.png.”



```
lea     rax, aCWindowsIconPn ; "C:\\Windows\\Icon.png"
mov     ebx, 19
call    os_Stat          ; check fo the icon.png file
nop
test    rcx, rcx
jz      short icon_file_exists ; check for err
```

```
call    B0Client_CreateIcon_CreateIcon ; Create icon file
mov     rax, cs:B0Client_GetConfig_ExtensionExists
mov     rbx, cs:g_len_Extension_string
call    B0Client_SetIconWithExtention_SetIconForExtension ; set default icon for all files with the ransomware's specified extension
nop     dword ptr [rax]
```

```
icon_file_exists:
call    B0Client_StopImportantServices_StopImportantService
call    B0Client_StopImportantServices_StopIISAndDeleteLogs
call    B0Client_StopImportantServices_StopSQLServices
call    B0Client_KillProccess_KillProcess
```

Check for “Icon.png”, and create if not found.

Process and Service targeting

Before proceeding with key derivation and encryption, the ransomware will attempt to stop related services for Windows IIS, MSSQL, and MySQL by executing the commands listed below.

BlockListed services

- cmd /C iisreset stop
- cmd /C NET STOP IISADMIN
- cmd /C net stop WAS
- cmd /C NET stop MSSQLSERVER
- cmd /C NET stop \”SQL Server (MSSQLSERVER)\”
- cmd /C net stop MSSQL$SQLEXPRESS
- cmd /C net stop SQLSERVERAGENT
- cmd /C net stop mysql
- sc stop W3SVC
- sc stop WAS
- sc stop IISADMIN

It's important to note that not only it targets specific services by name but also attempts to obtain a list of running services and processes using the commands "sc query" and "tasklist." Additionally, it tries to terminate any service or process that contains the word "SQL" in its name.

Anti-forensics measures

The ransomware deploys a variety of anti-forensic measures, to avoid leaving behind any leads that can be used in a forensic investigation. The following is a list of the anti-forensics' measures taken by the ransomware to wipe any evidence.

1. Delete the Windows IIS logs located in the directory: **C:\inetpub\logs**.
2. Empty the Recycle Bin.
3. Constantly clears Windows Event Logs.
4. Constantly Clear the contents of the Windows Temp folder by running the following command:

```
Del /S /F /Q %Windir%\Temp
```

5. Create a registry entry under the Run key to clear Windows Event Logs upon user login. Use the following command:

```
powershell.exe -Command New-ItemProperty -Path
HKCU:\Software\Microsoft\Windows\CurrentVersion\Run -Name DeleteLogWithStartUp -
Value 'powershell wevtutil el | ForEach-Object { wevtutil cl $_ }' -PropertyType
String
```

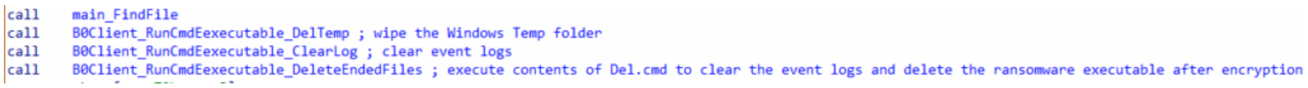
This will ensure that the Windows Event Logs are cleared each time the user logs in.

Furthermore, the following commands are written to a file called "Del.cmd," which is saved in the system directory C:\Windows\System32\ and executed. This script will delete the ransomware executable as well as the Windows Event logs.

Contents of Del.cmd, that will be executed after encryption to wipe any traces of its activity

```
timeout /t 10
del /f ransomware_executable_path
powershell "wevtutil el | Foreach-Object {wevtutil cl "$_"}
```

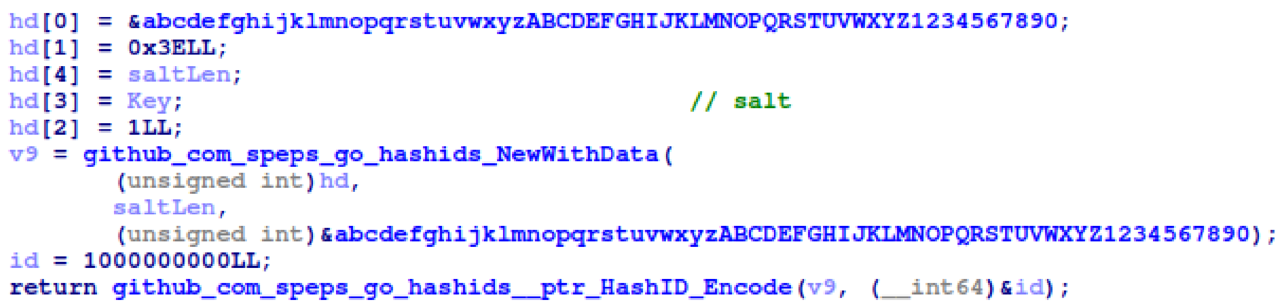
It is noteworthy to mention that the ransomware does not delete the Windows Event Logs; instead, it continuously clears them during execution to erase any traces of its activity. The same behavior applies to the Windows Temp folder.



```
call    main_FindFile
call    B0Client_RunCmdEexecutable_DelTemp ; wipe the Windows Temp folder
call    B0Client_RunCmdEexecutable_ClearLog ; clear event logs
call    B0Client_RunCmdEexecutable_DeleteEndedFiles ; execute contents of Del.cmd to clear the event logs and delete the ransomware executable after encryption
```

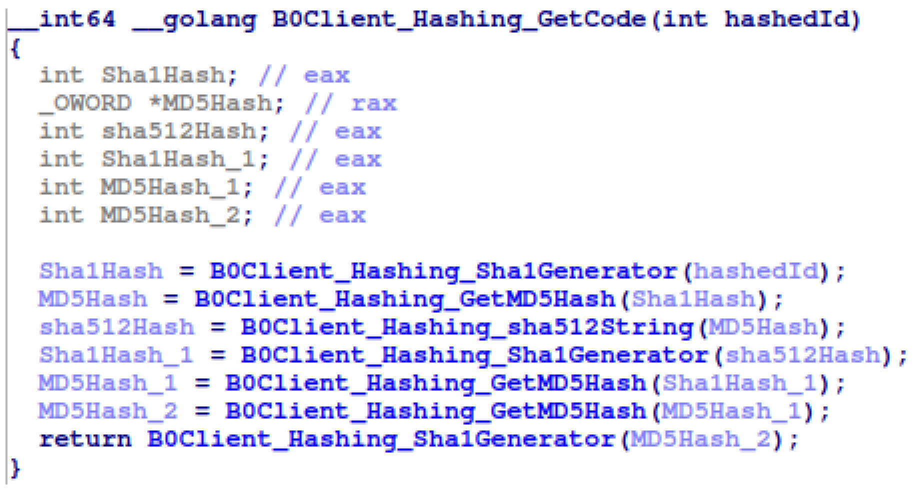
Key Derivation

The initially generated random nine-character string is used to generate the AES-GCM key, which is then used during encryption. The ransomware first uses the hashids library to generate a hashed ID using the random string as the salt and 1,000,000,000(0x3B9ACA00) . This library can generate hashed/obfuscated IDs from numbers. The hashed ID's size is 7 bytes, and its character set is limited to 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'.



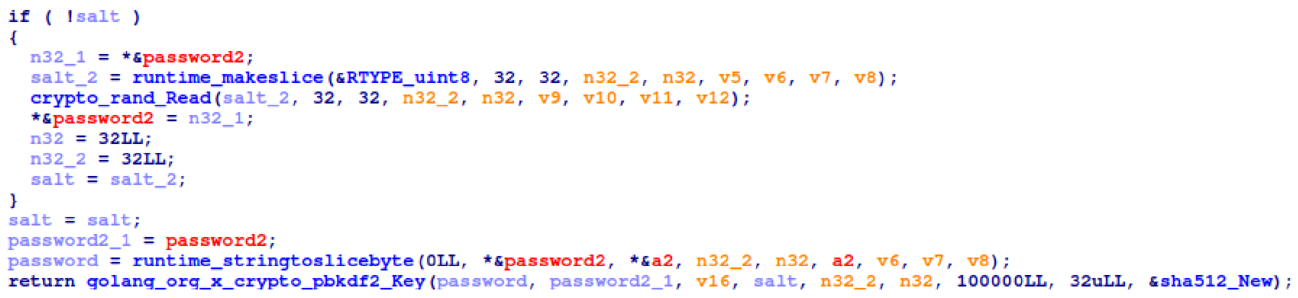
```
hd[0] = &abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890;
hd[1] = 0x3ELL;
hd[4] = saltLen;
hd[3] = Key;                                    // salt
hd[2] = 1LL;
v9 = github_com_speps_go_hashids_NewWithData(
       (unsigned int)hd,
       saltLen,
       (unsigned int)&abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890);
id = 1000000000LL;
return github_com_speps_go_hashids__ptr_HashID_Encode(v9, (__int64)&id);
```

The hashed ID is then hashed in the following order: SHA1 → MD5 → SHA512 → SHA1 → MD5 → MD5 → SHA1 to generate a password that will be used in the password based key derivation function. The hashing functions also encode the hashed data before returning it.



```
__int64 __golang B0Client_Hashing_GetCode(int hashedId)
{
  int Sha1Hash; // eax
  _OWORD *MD5Hash; // rax
  int sha512Hash; // eax
  int Sha1Hash_1; // eax
  int MD5Hash_1; // eax
  int MD5Hash_2; // eax

  Sha1Hash = B0Client_Hashing_Sha1Generator(hashedId);
  MD5Hash = B0Client_Hashing_GetMD5Hash(Sha1Hash);
  sha512Hash = B0Client_Hashing_sha512String(MD5Hash);
  Sha1Hash_1 = B0Client_Hashing_Sha1Generator(sha512Hash);
  MD5Hash_1 = B0Client_Hashing_GetMD5Hash(Sha1Hash_1);
  MD5Hash_2 = B0Client_Hashing_GetMD5Hash(MD5Hash_1);
  return B0Client_Hashing_Sha1Generator(MD5Hash_2);
}
```

After this, PBKDF2 is used to derive the AES-GCM key from the password. The salt is the same as the password; 100,000 iterations are used, and the key length is 32 bytes. SHA512 is used as the hash function for the password based key derivation function. Notably, if a null value is passed as the salt(that indicates for some reason it failed to generate and hash the random nine-character string), the ransomware generates a random salt using the crypto/rand package; however, if this occurs, it is impossible to decrypt the file even with the threat actor’s decryptor, since the ransomware does not store the salt somewhere. Regardless, this scenario would not happen, as the third argument is always the password converted to bytes in the current version of the ransomware.



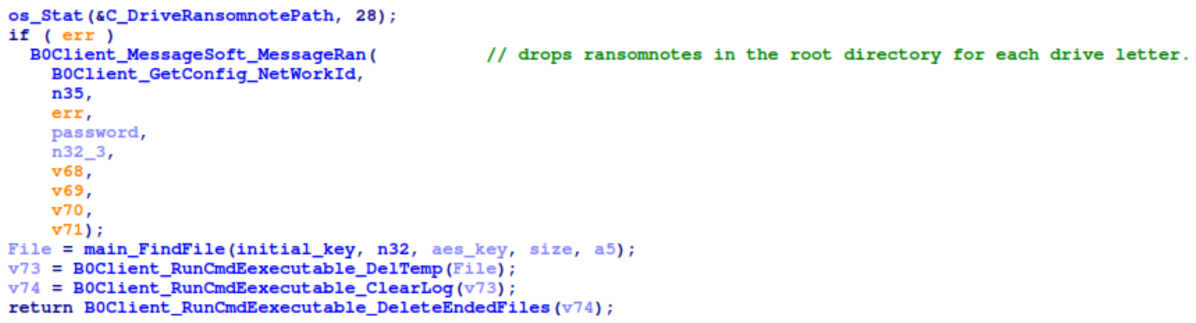
```
if ( !salt )
{
  n32_1 = *&password2;
  salt_2 = runtime_makeslice(&RTYPE_uint8, 32, 32, n32_2, n32, v5, v6, v7, v8);
  crypto_rand_Read(salt_2, 32, 32, n32_2, n32, v9, v10, v11, v12);
  *&password2 = n32_1;
  n32 = 32LL;
  n32_2 = 32LL;
  salt = salt_2;
}
salt = salt;
password2_1 = password2;
password = runtime_stringtoslicebyte(0LL, *&password2, *&a2, n32_2, n32, a2, v6, v7, v8);
return golang_org_x_crypto_pbkdf2_Key(password, password2_1, v16, salt, n32_2, n32, 100000LL, 32uLL, &sha512_New);
```

The resulting key is used to encrypt the files with AES-GCM, and the same key is used for each file.

Ransom Note

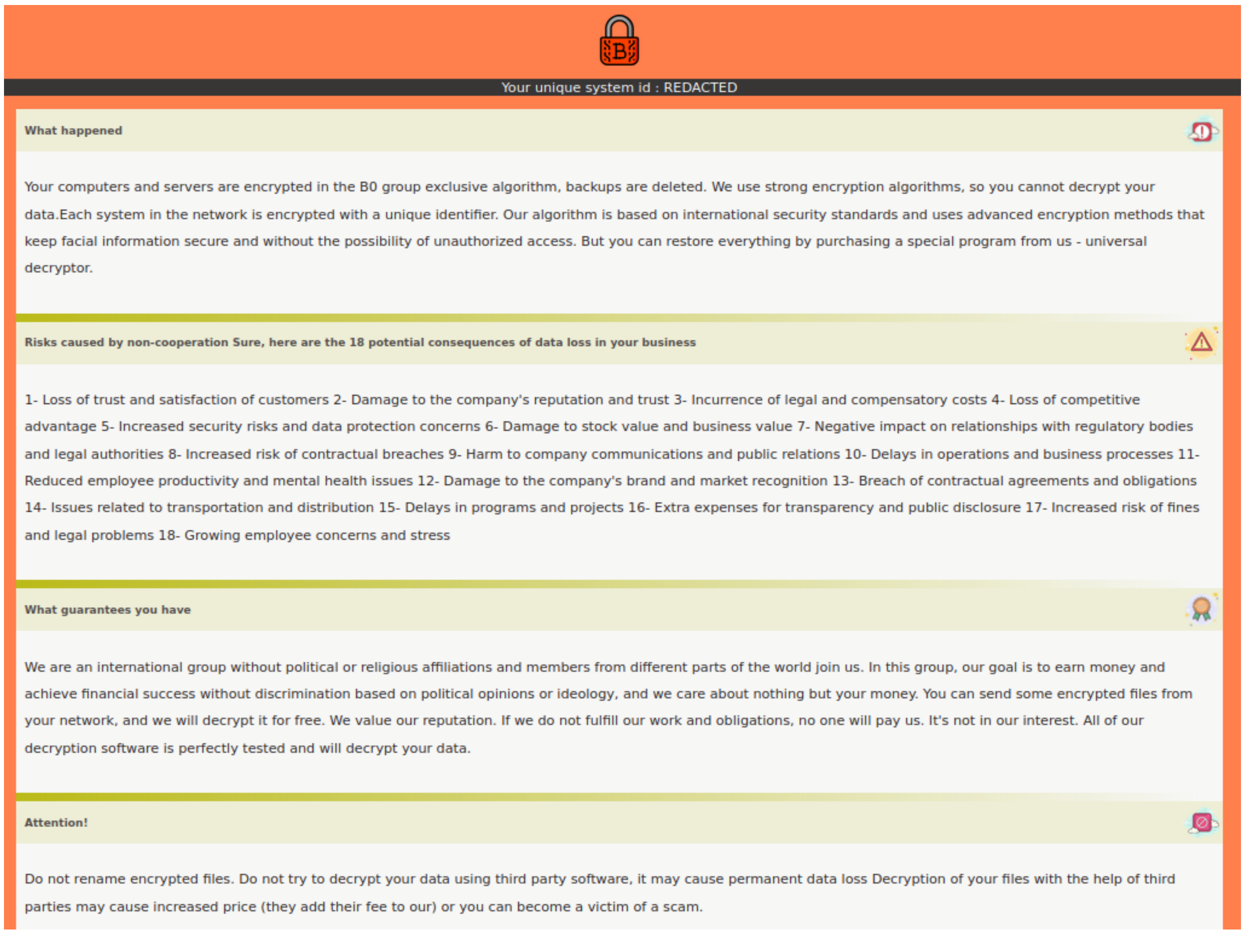
As is common to ransomware groups, a note with instructions on how to contact the actor is dropped in each driver’s root directory. The victim can negotiate payment options via a communication channel of the actor’s choosing. The note is dropped in two forms : a text file, How_To_Recovery_Files.txt, and an HTML file also named How_To_Recovery_Files.html with slightly different content.

It is worth mentioning that the Network ID(serves a unique identifier to the victim) provided earlier in the early of its execution, is added to the contents of the ransom note before it’s written to disk.



```
os_Stat(&C_DriveRansomnotePath, 28);
if ( err )
  B0Client_MessageSoft_MessageRan(          // drops ransomnotes in the root directory for each drive letter.
    B0Client_GetConfig_NetWorkId,
    n35,
    err,
    password,
    n32_3,
    v68,
    v69,
    v70,
    v71);
File = main_FindFile(initial_key, n32, aes_key, size, a5);
v73 = B0Client_RunCmdEexecutable_DelTemp(File);
v74 = B0Client_RunCmdEexecutable_ClearLog(v73);
return B0Client_RunCmdEexecutable_DeleteEndedFiles(v74);
```

How the ransom notes are dropped in the root directory of each drive letter.



Your unique system id : REDACTED

What happened

Your computers and servers are encrypted in the B0 group exclusive algorithm, backups are deleted. We use strong encryption algorithms, so you cannot decrypt your data.Each system in the network is encrypted with a unique identifier. Our algorithm is based on international security standards and uses advanced encryption methods that keep facial information secure and without the possibility of unauthorized access. But you can restore everything by purchasing a special program from us - universal decryptor.

Risks caused by non-cooperation Sure, here are the 18 potential consequences of data loss in your business

1- Loss of trust and satisfaction of customers 2- Damage to the company's reputation and trust 3- Incurrence of legal and compensatory costs 4- Loss of competitive advantage 5- Increased security risks and data protection concerns 6- Damage to stock value and business value 7- Negative impact on relationships with regulatory bodies and legal authorities 8- Increased risk of contractual breaches 9- Harm to company communications and public relations 10- Delays in operations and business processes 11- Reduced employee productivity and mental health issues 12- Damage to the company's brand and market recognition 13- Breach of contractual agreements and obligations 14- Issues related to transportation and distribution 15- Delays in programs and projects 16- Extra expenses for transparency and public disclosure 17- Increased risk of fines and legal problems 18- Growing employee concerns and stress

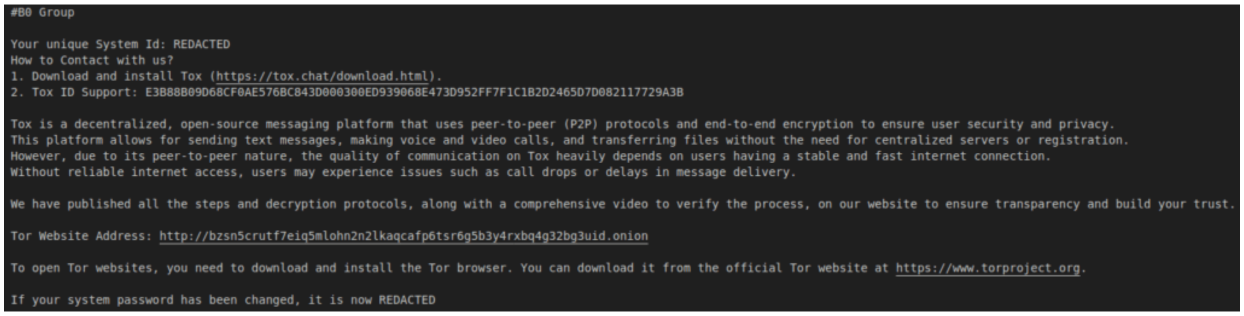
What guarantees you have

We are an international group without political or religious affiliations and members from different parts of the world join us. In this group, our goal is to earn money and achieve financial success without discrimination based on political opinions or ideology, and we care about nothing but your money. You can send some encrypted files from your network, and we will decrypt it for free. We value our reputation. If we do not fulfill our work and obligations, no one will pay us. It's not in our interest. All of our decryption software is perfectly tested and will decrypt your data.

Attention!

Do not rename encrypted files. Do not try to decrypt your data using third party software, it may cause permanent data loss Decryption of your files with the help of third parties may cause increased price (they add their fee to our) or you can become a victim of a scam.

HTML version of the ransomware note.

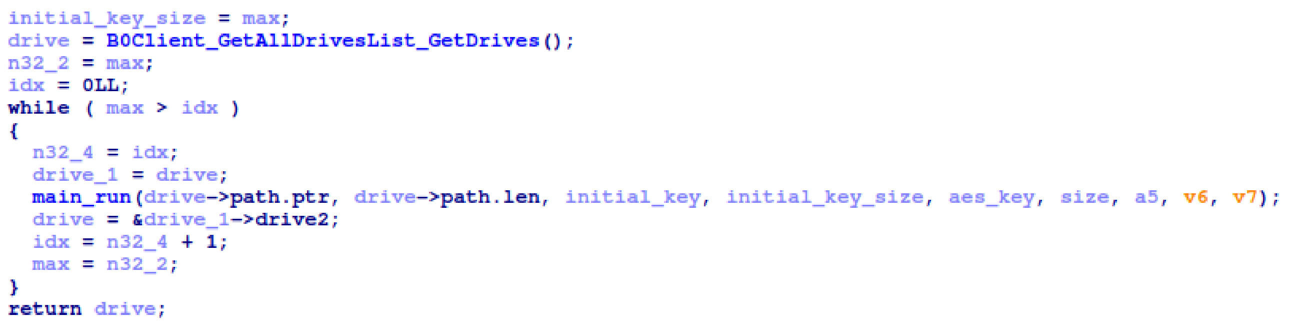


```
#B0 Group

Your unique System Id: REDACTED
How to Contact with us?
1. Download and install Tox (https://tox.chat/download.html).
2. Tox ID Support: E3B88B09D68CF0AE576BC843D000300ED939068E473D952FF7F1C1B2D2465D7D082117729A3B

Tox is a decentralized, open-source messaging platform that uses peer-to-peer (P2P) protocols and end-to-end encryption to ensure user security and privacy.
This platform allows for sending text messages, making voice and video calls, and transferring files without the need for centralized servers or registration.
However, due to its peer-to-peer nature, the quality of communication on Tox heavily depends on users having a stable and fast internet connection.
Without reliable internet access, users may experience issues such as call drops or delays in message delivery.

We have published all the steps and decryption protocols, along with a comprehensive video to verify the process, on our website to ensure transparency and build your trust.

Tor Website Address: http://bzsn5crutf7eiq5mlohn2n2lkaqcafp6tsr6g5b3y4rxbq4g32bg3uid.onion

To open Tor websites, you need to download and install the Tor browser. You can download it from the official Tor website at https://www.torproject.org.

If your system password has been changed, it is now REDACTED
```

Text version of the ransomware note.

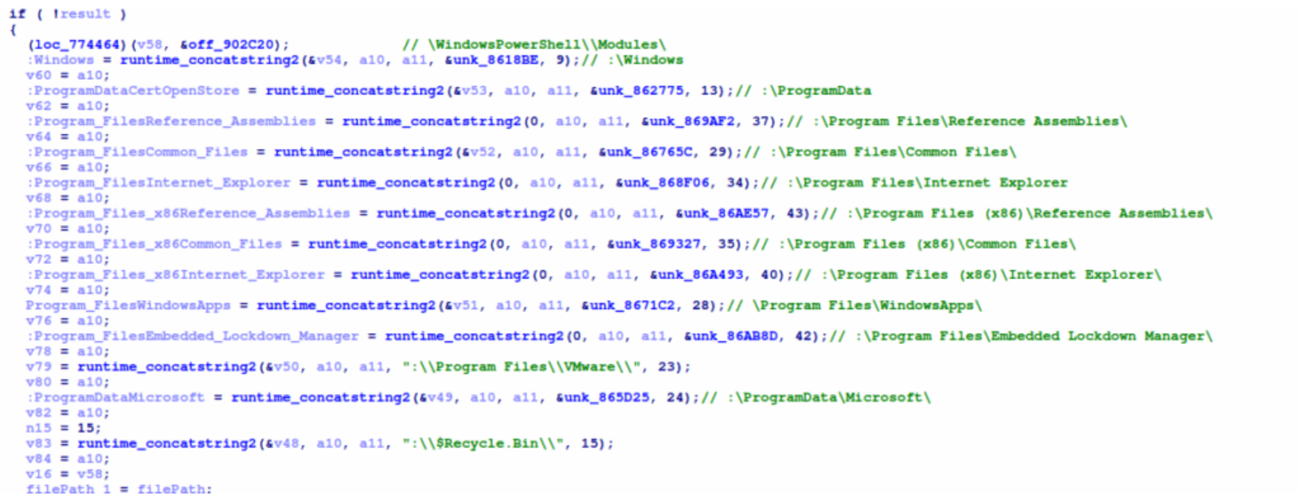
File Encryption

The *main_FindFile* function retrieves a list of the root directory paths for each available drive letter, and then calls *main_run* on each.



```
initial_key_size = max;
drive = BOClient_GetAllDrivesList_GetDrives();
n32_2 = max;
idx = 0LL;
while ( max > idx )
{
  n32_4 = idx;
  drive_1 = drive;
  main_run(drive->path.ptr, drive->path.len, initial_key, initial_key_size, aes_key, size, a5, v6, v7);
  drive = &drive_1->drive2;
  idx = n32_4 + 1;
  max = n32_2;
}
return drive;
```

The *main_run* uses *filepath.Walk()* to recursively walk through all the files, and directories. The callback function passed as an argument to *filepath.Walk()* performs some checks on the file path to decide whether the file can be encrypted or not. It first checks whether any of the following blacklisted paths are in the filepath, and if they are, the callback function returns without encrypting the file.



```
if ( !result )
{
  (loc_774464)(v58, &off_902C20);              // \WindowsPowerShell\\Modules\
  :Windows = runtime_concatstring2(&v54, a10, a11, &unk_8618BE, 9);// :\Windows
  v60 = a10;
  :ProgramDataCertOpenStore = runtime_concatstring2(&v53, a10, a11, &unk_862775, 13);// :\ProgramData
  v62 = a10;
  :Program_FilesReference_Assemblies = runtime_concatstring2(0, a10, a11, &unk_869AF2, 37);// :\Program Files\Reference Assemblies\
  v64 = a10;
  :Program_FilesCommon_Files = runtime_concatstring2(&v52, a10, a11, &unk_86765C, 29);// :\Program Files\Common Files\
  v66 = a10;
  :Program_FilesInternet_Explorer = runtime_concatstring2(0, a10, a11, &unk_868F06, 34);// :\Program Files\Internet Explorer
  v68 = a10;
  :Program_Files_x86Reference_Assemblies = runtime_concatstring2(0, a10, a11, &unk_86AE57, 43);// :\Program Files (x86)\Reference Assemblies\
  v70 = a10;
  :Program_Files_x86Common_Files = runtime_concatstring2(0, a10, a11, &unk_869327, 35);// :\Program Files (x86)\Common Files\
  v72 = a10;
  :Program_Files_x86Internet_Explorer = runtime_concatstring2(0, a10, a11, &unk_86A493, 40);// :\Program Files (x86)\Internet Explorer\
  v74 = a10;
  Program_FilesWindowsApps = runtime_concatstring2(&v51, a10, a11, &unk_8671C2, 28);// \Program Files\WindowsApps\
  v76 = a10;
  :Program_FilesEmbedded_Lockdown_Manager = runtime_concatstring2(0, a10, a11, &unk_86AB8D, 42);// :\Program Files\Embedded Lockdown Manager\
  v78 = a10;
  v79 = runtime_concatstring2(&v50, a10, a11, ":\\Program Files\\VMware\\", 23);
  v80 = a10;
  :ProgramDataMicrosoft = runtime_concatstring2(&v49, a10, a11, &unk_865D25, 24);// :\ProgramData\Microsoft\
  v82 = a10;
  n15 = 15;
  v83 = runtime_concatstring2(&v48, a10, a11, ":\\$Recycle.Bin\\", 15);
  v84 = a10;
  v16 = v58;
  filePath_1 = filePath;
```

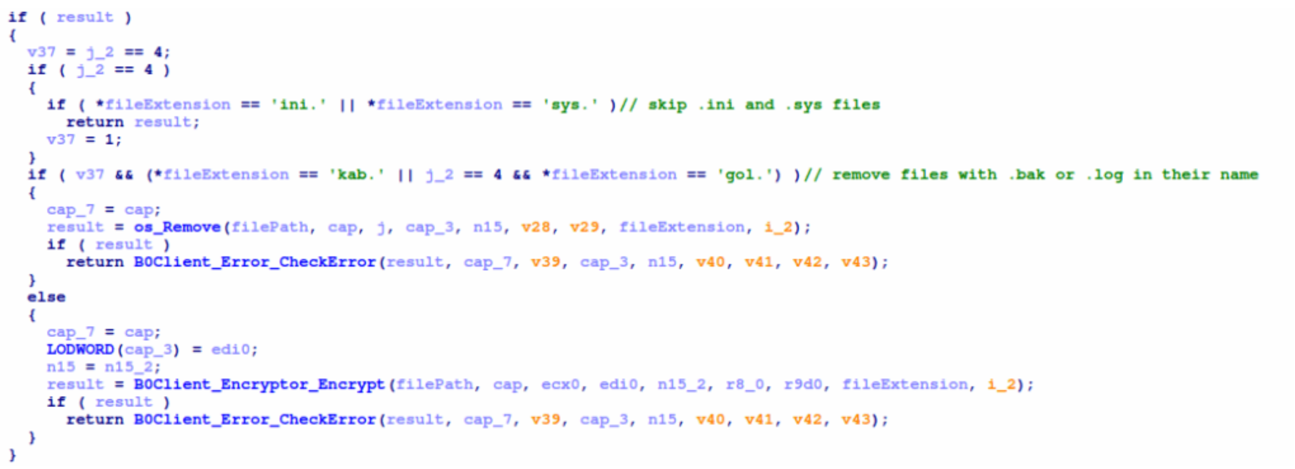
Below we summarize the list of paths excluded by the ransomware during encryption.

BlackListed Filepaths:

- \WindowsPowerShell\Modules\
- \Windows
- \ProgramData
- \Program Files\Reference Assemblies\
- \Program Files\Common Files\
- \Program Files\Internet Explorer
- \Program Files (x86)\Reference Assemblies\
- \Program Files (x86)\Common Files\
- \Program Files (x86)\Internet Explorer\
- \Program Files\WindowsApps\
- \Program Files\Embedded Lockdown Manager\
- \Program Files\VMware\
- \ProgramData\Microsoft\

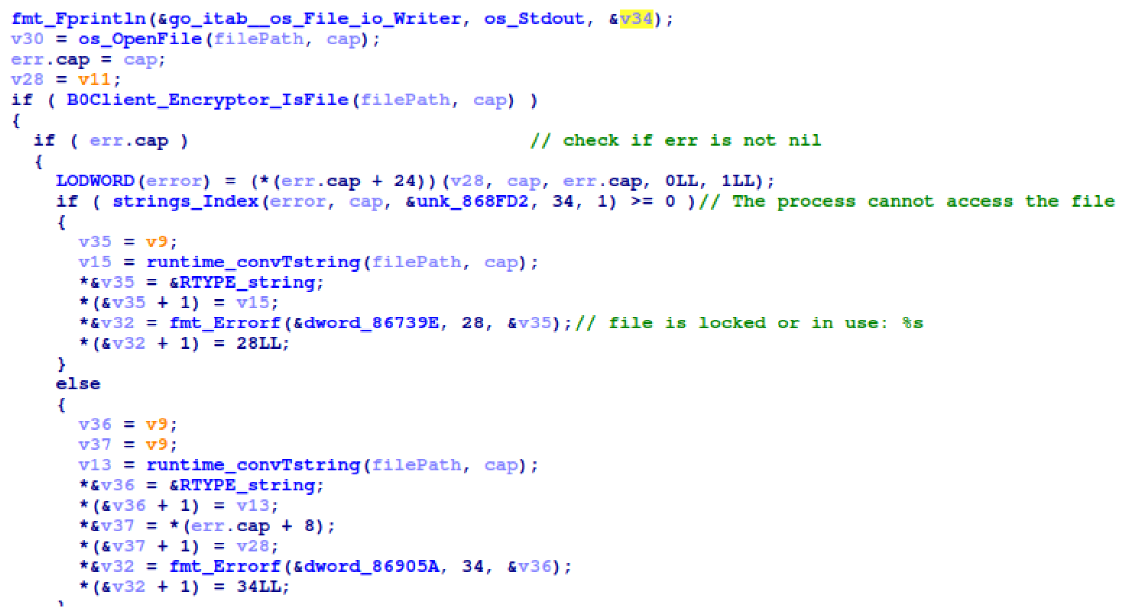
- \$Recycle.Bin\

The next check determines whether the ransomware extension is at the end of the file, and whether the extension is *.ini* or *.sys.* If it is either of these, the file is skipped. After this, it checks whether the extension is *.bak* or *.log*, and if it is, it proceeds with deleting the file and returning. If a file doesn't pass any of these checks, the encryption function is called on it. Since these checks do not check whether the file is the ransomware executable itself, the ransomware ends up encrypting its own executable. However, it fails to remove the plaintext executable after the encryption process using *os.remove*.



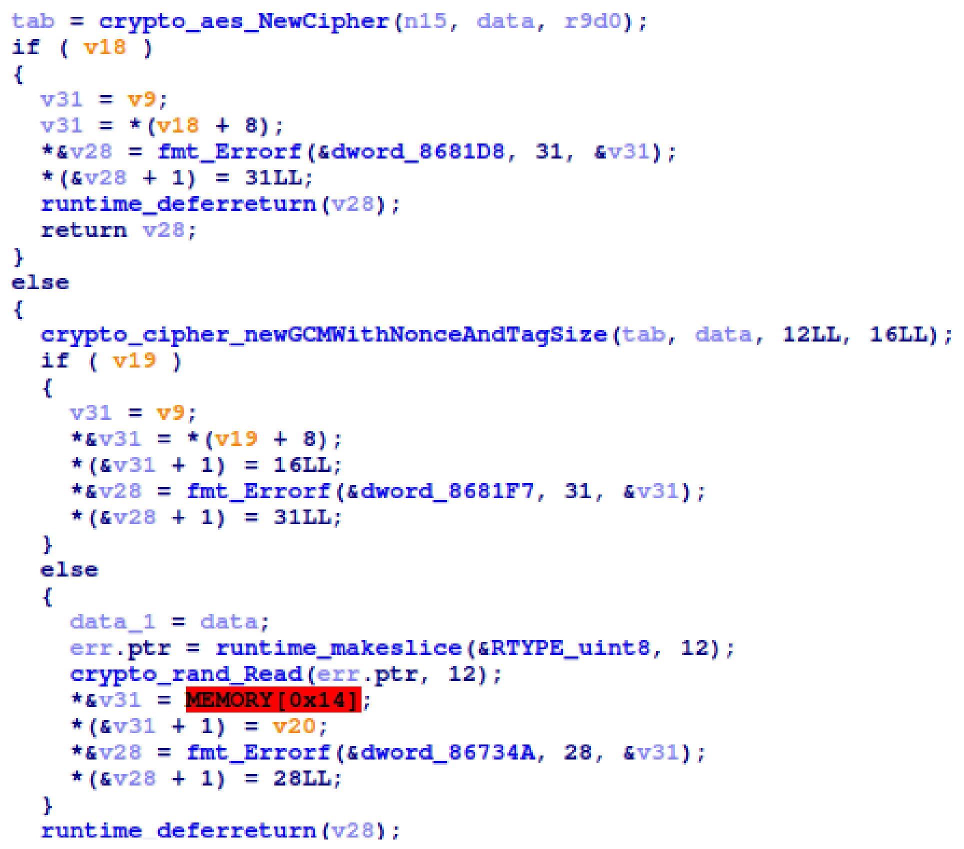
```
if ( result )
{
  v37 = j_2 == 4;
  if ( j_2 == 4 )
  {
    if ( *fileExtension == 'ini.' || *fileExtension == 'sys.' )// skip .ini and .sys files
      return result;
    v37 = 1;
  }
  if ( v37 && (*fileExtension == 'kab.' || j_2 == 4 && *fileExtension == 'gol.') )// remove files with .bak or .log in their name
  {
    cap_7 = cap;
    result = os_Remove(filePath, cap, j, cap_3, n15, v28, v29, fileExtension, i_2);
    if ( result )
      return BOClient_Error_CheckError(result, cap_7, v39, cap_3, n15, v40, v41, v42, v43);
  }
  else
  {
    cap_7 = cap;
    LODWORD(cap_3) = edi0;
    n15 = n15_2;
    result = BOClient_Encryptor_Encrypt(filePath, cap, ecx0, edi0, n15_2, r8_0, r9d0, fileExtension, i_2);
    if ( result )
      return BOClient_Error_CheckError(result, cap_7, v39, cap_3, n15, v40, v41, v42, v43);
  }
}
```

The file encryption function first prints the filepath. After which, it attempts to open the file. If that fails, the ransomware just skips the file instead of trying to kill the process that's using the file. The ransomware also checks that the file is not a directory after opening it.



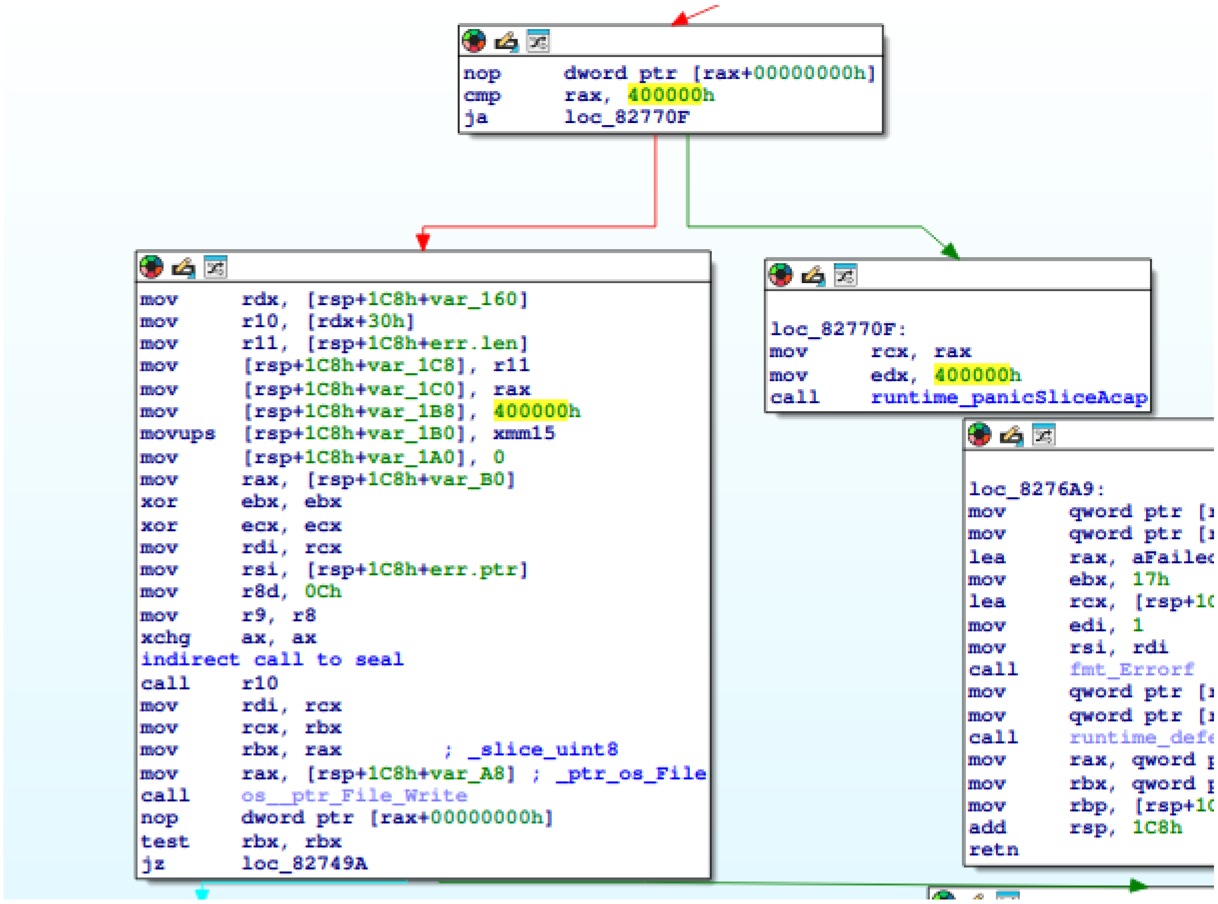
```
fmt_Fprintln(&go_itab__os_File_io_Writer, os_Stdout, &v34);
v30 = os_OpenFile(filePath, cap);
err.cap = cap;
v28 = v11;
if ( BOClient_Encryptor_IsFile(filePath, cap) )
{
  if ( err.cap )                              // check if err is not nil
  {
    LODWORD(error) = (*(err.cap + 24))(v28, cap, err.cap, 0LL, 1LL);
    if ( strings_Index(error, cap, &unk_868FD2, 34, 1) >= 0 )// The process cannot access the file
    {
      v35 = v9;
      v15 = runtime_convTstring(filePath, cap);
      *&v35 = &RTYPE_string;
      *(&v35 + 1) = v15;
      *&v32 = fmt_Errorf(&dword_86739E, 28, &v35);// file is locked or in use: %s
      *(&v32 + 1) = 28LL;
    }
    else
    {
      v36 = v9;
      v37 = v9;
      v13 = runtime_convTstring(filePath, cap);
      *&v36 = &RTYPE_string;
      *(&v36 + 1) = v13;
      *&v37 = *(err.cap + 8);
      *(&v37 + 1) = v28;
      *&v32 = fmt_Errorf(&dword_86905A, 34, &v36);
      *(&v32 + 1) = 34LL;
```

The ransomware initiates the encryption process by generating an AES cipher object and initializing it with the previously generated key. It uses GCM mode with a 16-byte tag size and a 12-byte nonce size. The nonce is randomly generated using the *crypto/rand* package.



```
tab = crypto_aes_NewCipher(n15, data, r9d0);
if ( v18 )
{
  v31 = v9;
  v31 = *(v18 + 8);
  *&v28 = fmt_Errorf(&dword_8681D8, 31, &v31);
  *(&v28 + 1) = 31LL;
  runtime_deferreturn(v28);
  return v28;
}
else
{
  crypto_cipher_newGCMWithNonceAndTagSize(tab, data, 12LL, 16LL);
  if ( v19 )
  {
    v31 = v9;
    *&v31 = *(v19 + 8);
    *(&v31 + 1) = 16LL;
    *&v28 = fmt_Errorf(&dword_8681F7, 31, &v31);
    *(&v28 + 1) = 31LL;
  }
  else
  {
    data_1 = data;
    err.ptr = runtime_makeslice(&RTYPE_uint8, 12);
    crypto_rand_Read(err.ptr, 12);
    *&v31 = MEMORY[0x14];
    *(&v31 + 1) = v20;
    *&v28 = fmt_Errorf(&dword_86734A, 28, &v31);
    *(&v28 + 1) = 28LL;
  }
  runtime_deferreturn(v28);
```

After this, the ransomware creates a new file named by appending *.id-*. to the original filename. The initially generated key is stored both in the configuration file and within each encrypted file's name. The ransomware then writes a 16-byte marker, consisting of null bytes, followed by the random nonce, to the new file. After this, it reads data from the original file in chunks of 0x400000 bytes, encrypts each chunk using the *Seal()* function, which handles both encryption and authentication. The resulting ciphertext, along with the 16-byte authentication tag, are both written to the new file. Once all chunks have been encrypted and written, the original file is deleted using *os.remove*.



```
nop     dword ptr [rax+00000000h]
cmp     rax, 400000h
ja      loc_82770F
```

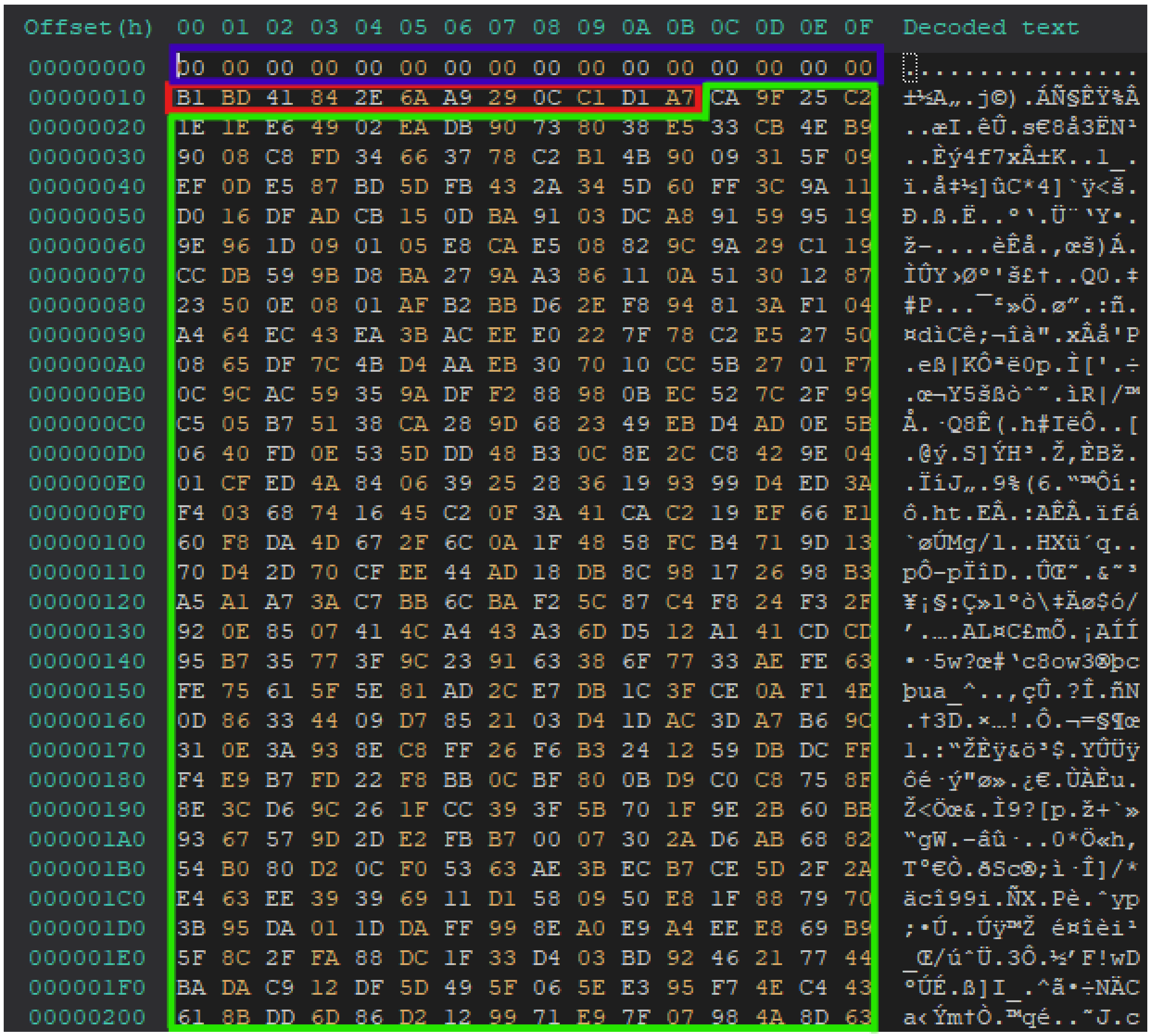
```
mov     rdx, [rsp+1C8h+var_160]
mov     r10, [rdx+30h]
mov     r11, [rsp+1C8h+err.len]
mov     [rsp+1C8h+var_1C8], r11
mov     [rsp+1C8h+var_1C0], rax
mov     [rsp+1C8h+var_1B8], 400000h
movups  [rsp+1C8h+var_1B0], xmm15
mov     [rsp+1C8h+var_1A0], 0
mov     rax, [rsp+1C8h+var_B0]
xor     ebx, ebx
xor     ecx, ecx
mov     rdi, rcx
mov     rsi, [rsp+1C8h+err.ptr]
mov     r8d, 0Ch
mov     r9, r8
xchg    ax, ax
indirect call to seal
call    r10
mov     rdi, rcx
mov     rcx, rbx
mov     rbx, rax          ; _slice_uint8
mov     rax, [rsp+1C8h+var_A8] ; _ptr_os_File
call    os__ptr_File_Write
nop     dword ptr [rax+00000000h]
test    rbx, rbx
jz      loc_82749A
```

```
loc_82770F:
mov     rcx, rax
mov     edx, 400000h
call    runtime_panicSliceAcap
```

```
loc_8276A9:
mov     qword ptr [
mov     qword ptr [
lea     rax, aFailed
mov     ebx, 17h
lea     rcx, [rsp+1C
mov     edi, 1
mov     rsi, rdi
call    fmt_Errorf
mov     qword ptr [
mov     qword ptr [
call    runtime_defe
mov     rax, qword p
mov     rbx, qword p
mov     rbp, [rsp+1C
add     rsp, 1C8h
retn
```

Encrypted file structure

The structure of the encrypted file is quite straightforward. At the beginning of the encryption process, a 16-byte marker (highlighted in purple) consisting of NULL bytes is added to the start of the file to indicate that it is encrypted. Following this marker, there is a 12-byte nonce (highlighted in Red in the screenshot below) value that remains consistent throughout the entire file. This nonce is stored within the file for use during decryption later. The rest of the data is just the file's encrypted data (highlighted in light green in the screenshot below).



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000010  B1 BD 41 84 2E 6A A9 29 0C C1 D1 A7 CA 9F 25 C2  ±½A„.j©).ÁÑ§ÊŸ%Â
00000020  1E 1E E6 49 02 EA DB 90 73 80 38 E5 33 CB 4E B9  ..æI.êÛ.s€8å3ËN¹
00000030  90 08 C8 FD 34 66 37 78 C2 B1 4B 90 09 31 5F 09  ..Èý4f7xÂ±K..1_.
00000040  EF 0D E5 87 BD 5D FB 43 2A 34 5D 60 FF 3C 9A 11  ï.å‡½]ûC*4]`ÿ<š.
00000050  D0 16 DF AD CB 15 0D BA 91 03 DC A8 91 59 95 19  Ð.ß.Ë..º'.Ü¨'Y•.
00000060  9E 96 1D 09 01 05 E8 CA E5 08 82 9C 9A 29 C1 19  ž–....èÊå.‚œš)Á.
00000070  CC DB 59 9B D8 BA 27 9A A3 86 11 0A 51 30 12 87  ÌÛY›Øº'š£†..Q0.‡
00000080  23 50 0E 08 01 AF B2 BB D6 2E F8 94 81 3A F1 04  #P...¯²»Ö.ø".:ñ.
00000090  A4 64 EC 43 EA 3B AC EE E0 22 7F 78 C2 E5 27 50  ¤dìCê;¬îà".xÂå'P
000000A0  08 65 DF 7C 4B D4 AA EB 30 70 10 CC 5B 27 01 F7  .eß|KÔªë0p.Ì['.÷
000000B0  0C 9C AC 59 35 9A DF F2 88 98 0B EC 52 7C 2F 99  .œ¬Y5šßòˆ˜.ìR|/™
000000C0  C5 05 B7 51 38 CA 28 9D 68 23 49 EB D4 AD 0E 5B  Å.·Q8Ê(.h#IëÔ..[
000000D0  06 40 FD 0E 53 5D DD 48 B3 0C 8E 2C C8 42 9E 04  .@ý.S]ÝH³.Ž,ÈBž.
000000E0  01 CF ED 4A 84 06 39 25 28 36 19 93 99 D4 ED 3A  .ÏíJ„.9%(6."™Ôí:
000000F0  F4 03 68 74 16 45 C2 0F 3A 41 CA C2 19 EF 66 E1  ô.ht.EÂ.:AÊÂ.ïfá
00000100  60 F8 DA 4D 67 2F 6C 0A 1F 48 58 FC B4 71 9D 13  `øÚMg/l..HXü´q..
00000110  70 D4 2D 70 CF EE 44 AD 18 DB 8C 98 17 26 98 B3  pÔ-pÏîD..ÛŒ˜.&˜³
00000120  A5 A1 A7 3A C7 BB 6C BA F2 5C 87 C4 F8 24 F3 2F  ¥¡§:Ç»lºò\‡Äø$ó/
00000130  92 0E 85 07 41 4C A4 43 A3 6D D5 12 A1 41 CD CD  '.….AL¤C£mÕ.¡AÍÍ
00000140  95 B7 35 77 3F 9C 23 91 63 38 6F 77 33 AE FE 63  •·5w?œ#'c8ow3®þc
00000150  FE 75 61 5F 5E 81 AD 2C E7 DB 1C 3F CE 0A F1 4E  þua_^..,çÛ.?Î.ñN
00000160  0D 86 33 44 09 D7 85 21 03 D4 1D AC 3D A7 B6 9C  .†3D.×…!.Ô.¬=§¶œ
00000170  31 0E 3A 93 8E C8 FF 26 F6 B3 24 12 59 DB DC FF  1.:"ŽÈÿ&ö³$.YÛÜÿ
00000180  F4 E9 B7 FD 22 F8 BB 0C BF 80 0B D9 C0 C8 75 8F  ôé·ý"ø».¿€.ÙÀÈu.
00000190  8E 3C D6 9C 26 1F CC 39 3F 5B 70 1F 9E 2B 60 BB  Ž<Öœ&.Ì9?[p.ž+`»
000001A0  93 67 57 9D 2D E2 FB B7 00 07 30 2A D6 AB 68 82  "gW.-âû·..0*Ö«h‚
000001B0  54 B0 80 D2 0C F0 53 63 AE 3B EC B7 CE 5D 2F 2A  T°€Ò.ðSc®;ì·Î]/*
000001C0  E4 63 EE 39 39 69 11 D1 58 09 50 E8 1F 88 79 70  äcî99i.ÑX.Pè.ˆyp
000001D0  3B 95 DA 01 1D DA FF 99 8E A0 E9 A4 EE E8 69 B9  ;•Ú..Úÿ™Ž é¤îèi¹
000001E0  5F 8C 2F FA 88 DC 1F 33 D4 03 BD 92 46 21 77 44  _Œ/úˆÜ.3Ô.½'F!wD
000001F0  BA DA C9 12 DF 5D 49 5F 06 5E E3 95 F7 4E C4 43  ºÚÉ.ß]I_.^ã•÷NÄC
00000200  61 8B DD 6D 86 D2 12 99 71 E9 7F 07 98 4A 8D 63  a‹Ým†Ò.™qé..˜J.c
```

Encrypted file structure highlighting marker, nonce, and encrypted bytes.

Weakness

There are two major weaknesses in the encryption. The first, and most obvious, is that the AES-GCM key can be easily derived from the initial key (randomly generated nine-character string), which is unique to each victim and generated at run-time, then saved to disk later.

The second weakness is less apparent. Although the ransomware uses AES-GCM to encrypt blocks (which combines AES-CTR for encryption and GMAC for authentication), it doesn't generate a new nonce for each block. As a result, the same keystream is reused for each block within a file. If we know the plaintext for one block, we can recover the keystream by xoring it with the corresponding ciphertext and then use that keystream to decrypt all subsequent blocks. However, exploiting this vulnerability is not straightforward, since it requires a significant amount of known plaintext (0x400000) bytes.

Decryptor

We have created a b0 ransomware decryptor for those affected by this ransomware. Errors are logged in .\log.txt. Below is the usage guide.

Usage of decryptor:

- dirpath string: Path to the directory to recursively decrypt files
- key string: Key to use for decryption, in the following filename test.pdf.id-QVKGBICKS.B0-3e72d, QVKGBICKS is the key.
- path string: Path to the encrypted file

Indicators Of Compromise

Known hashes

cea74b854b6cd161884e5ced5c8b7ba44c1064ff22703667cc913d9a67f1767b (SHA-256)

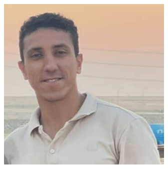
File Paths

Ransomware executable	B0.exe
Ransomware configuration written to disk at run-time	C:\Windows\System32\ telemetry.ASM-WindowsDefault.json
Ransomware icon	C:\Windows\Icon.png
Batch script	C:\Windows\System32\Del.cmd
Ransom note in text and HTML versions	C:\How_To_Recovery_Files.txt C:\How_To_Recovery_Files.html
Registry entry created to clear Windows Event logs on user login	HKCU:\Software\Microsoft\Windows\CurrentVersion\Run\DeleteLogWithStartUp

Conclusion

This investigation is a powerful reminder that even in an ecosystem dominated by rapidly evolving ransomware threats, there’s still room for skilled analysis, persistence, and creative problem-solving to make a real impact. While not every case ends in a decryptor, this one proves that deep malware analysis and DFIR expertise can tip the scales.

Authors

- 
  [Mohamed Talaat]
  Mohamed Talaat is a self-taught security researcher specializing in malware analysis, threat intelligence, and tracking APT groups and crimeware. With a background in engineering and over three years of independent research, his focus is on ransomware recovery, evaluating recovery paths, analyzing encryption schemes, and occasionally developing decryptors. Passionate about reverse engineering and low-level analysis, he also writes YARA rules to help detect and hunt emerging threats and campaigns.
-
  [Hassan Faraz]
  Specializing in malware reverse engineering and data recovery, Hassan Faraz develops decryption solutions for vulnerable ransomware and creates custom tools to extract data from various file formats such as VHD, VMDK, MDF, backup formats, etc. He has a proven track record of recovering critical file formats with minimal data loss, helping organizations mitigate the impact of cyberattacks.