Reverse Engineering

COURSE WORK TWO

COURSE ID: KH6052CEM

INSTRUCTOR: DR. AHMED SELIM

NAME: AHMED FAROUK MAHMOUD

STUDENT ID: CU2000512

Contents

List of Figures

List of Tables

Section 1: Executive Summary

SHA-256 hash	2FAA2637FEDC5788A9598691581000BA637DB86C8505FDE1DC9D7DBCA8CC41CD

The sample PE is a ransomware first identified by cybersecurity researchers in April 2019. This Ransomware belongs to the ransom group Sodinokibi, also known as REvil, it utilizes a combination of encryption and extortion tactics to disrupt normal device and data access. It specifically targets and encrypts files on a victim's disk and demands payment for a decryption key. The ransomware is primarily spread through human-operated campaigns via phishing emails containing malicious attachments or links, as well as exploiting vulnerabilities in various network services such as web browsers and VPN appliances. Once the attackers gain access to the network, they are known to steal login credentials, elevate their privileges, and propagate laterally to establish a persistent foothold before deploying the ransomware payload.

YARA signature rules are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

Basic Information	
File Name	Sample.exe
Malware Family	Sodinokibi, also known as REvil
Md5 hash	61c19e7ce627da9b5004371f867a47d3
Sha-256	bf7114f025fff7dbc6b7aff8e4edb0dd8a7b53c3766429a3c5f10142609968f9
File type	Win32 EXE
Target Machine	Windows operating systems 32/64 bits
Is packed	TRUE

Table 1 Basic Information

Section 2: Basic Static Analysis

2.1 Basic Analysis & File Header Analysis:

Properties	Values
File Type	Portable Executable (PE)
File can be executed	True
Compiler stamp	15/11/2018
Sections	SECTION UPX 0, SECTION UPX1
Entropy	5.35
Virtual Size & Raw Data	3584 bits more in virtual size than raw data (maybe packed malware)
Entry Point	004050ed0
Is packed	After analyzing the properties above I concluded that **this malware is packed**

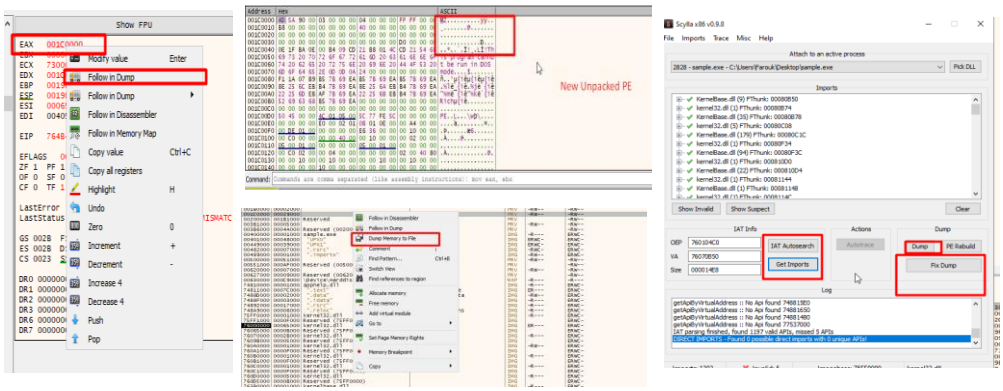
Table 2 Basic Analysis & File Header Analysis

2.2 Unpacking the Malware Sample:

As mentioned above I was able to identify that the malware sample was packed and, in this section, I demonstrated the unpacking process.

1. **Unpacking with UPX:** the section names where UPX 0 and UPX 1 which made me believe that the malware is packed using UPX but after trying to unpack it with UPX it failed, so I needed to unpack the sample manually using x32debug.
2. **Adding breakpoints:** I added a breakpoint at the return address from function virtual allocate.

3. **Follow the EAX in dump:** the EAX registry holds the new memory address, and I followed it in the dump and dump the memory to a new PE file.
4. **Fix the address table:** Load the new PE in x32 debugger and fix the import table with Scylla.



2.3 Extracted Strings After Unpacking:

Some of the interesting strings found in the portable executable where					
United States	United Kingdom	Spanish	French	October	Wednesday
New []	Delete []	Sin/Cos	To many files open in system		Write/move
ABCDEFGHIJKLMNOPQRSTUVWXYZ			Connected, not a socket, already connected		

Table 3 Extracted Strings After Unpacking

2.4 Imported Libraries:

Some of the imported Libraries				
user32.dll	ntdll.dll	ole32.dll	advapi32.dll	kernel32.dll
winhttp.dll	Crypt32.dll	Mpr.dll	Advapi32.dll	Shlwapi.dll

Table 4 Imported Libraries

2.5 Used functions:

Some of the interesting functions used in the portable executable				
wsprintfW	GetDiskFreeSpaceExW	CryptBinaryToStringW	WinHttpSendRequest	DeleteFileW
GetFileSize	RtlInitUnicodeString	CryptAcquireContextW	MoveFileW	WriteFile
Sleep	GetStockObject	ShellExecuteExW	WinHttpConnect	Sleep

Table 5 Interesting Functions

2.6 Summary of findings (Basic Static Analysis)

It is difficult to determine the exact capabilities of this sample file based solely on the basic static analysis, but it is possible that the sample file can perform malicious tasks such as:

Function Names	**Function capabilities**
DeleteFileW, CryptAcquireContextW, CreateFileMappingW, DeleteCriticalSection.	Could be used for various types of files and registry manipulation
GetDiskFreeSpaceExW and GetFileSize.	Could be used to gather information about the system and files on it
GetFileSize. CryptBinaryToStringW.	Could be used to encode data for transmission or storage.
GetDiskFreeSpaceExW, GetFileSize. CryptBinaryToStringW.	Could be used to encode data for transmission or storage.
timeGetTime and Sleep.	Could be used to slow down the execution of malware or to evade detection.

Table 6 Summary of findings (Basic Static Analysis)

Again, Basic static analysis of a Portable Executable (PE) file involves examining the file's properties and structure but does not involve executing the code in the file. As a result, it can be difficult to identify exactly what a PE file does based on static analysis alone.

Basic Static Analysis screenshots

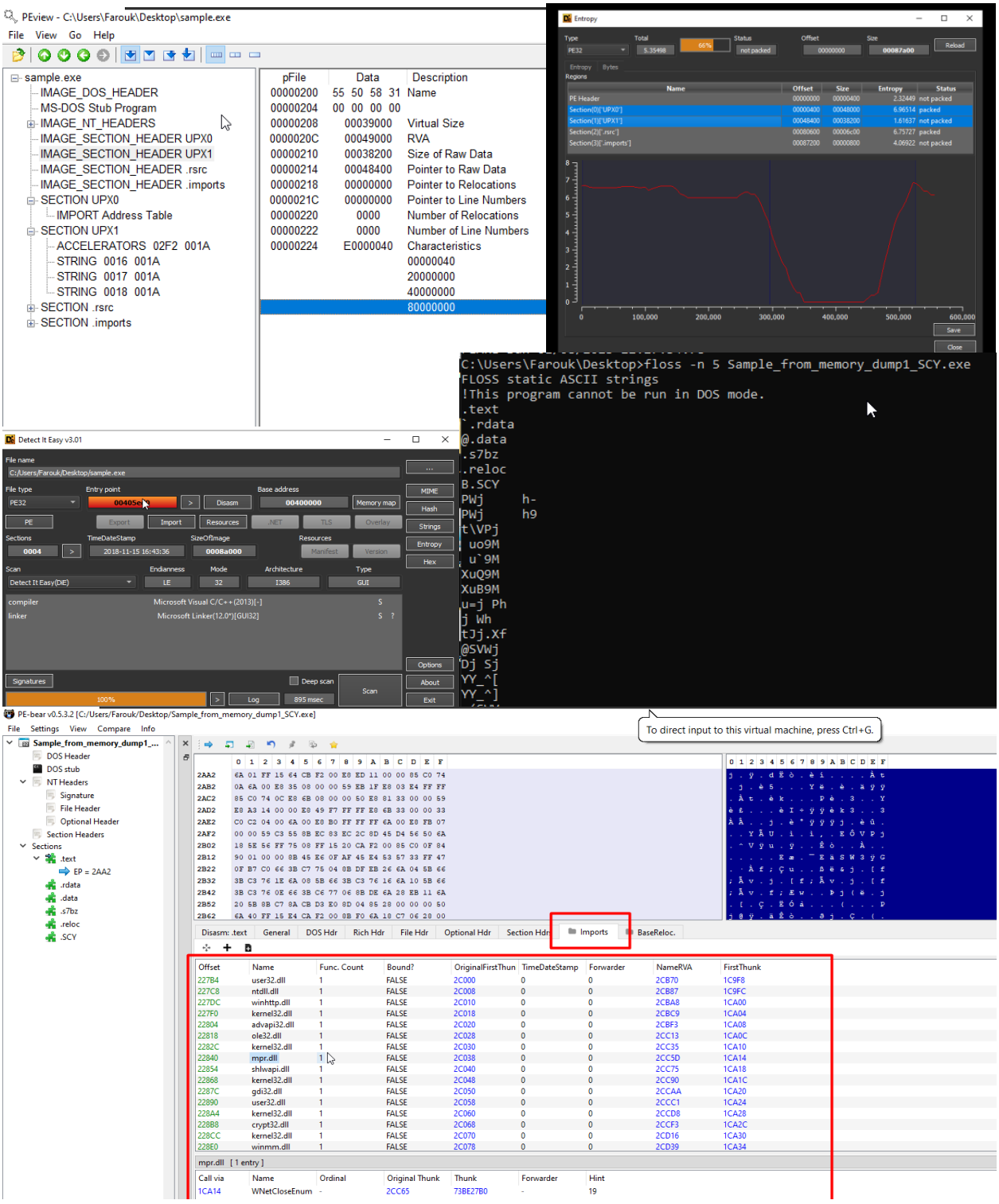
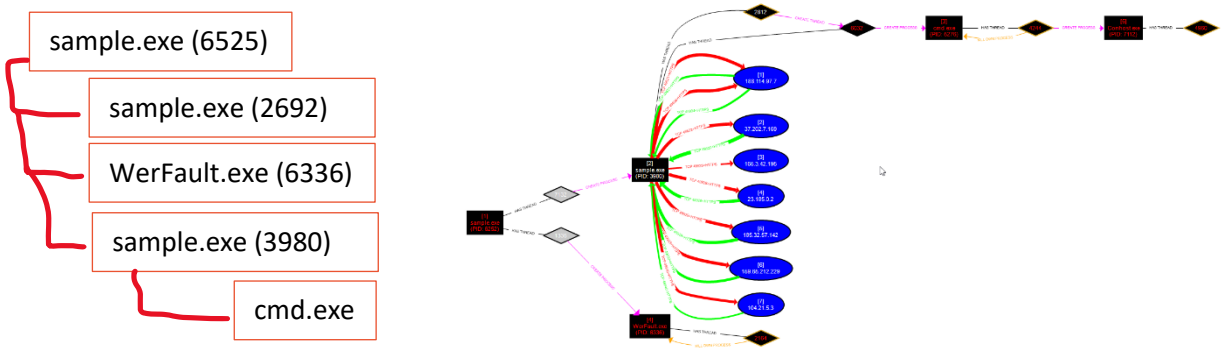


Figure 1 Basic Static Analysis screenshots

Section 3: Basic Dynamic Analysis

3.1 Process Monitoring

After running Procmon to monitor the processes and using Procdot to visualize the sample's activity, I was able to understand that the sample.exe started 3 other processes.



- The sample.exe encrypts all the files on the system
- Cmd.exe command executed: " /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default} recoveryenabled No & bcdedit /set {default} bootstatuspolicy ignoreallfailures
    - This command appears to contain several different actions separated by the & symbol. Together, these commands delete all shadow copies on the system, disable recovery mode, and set the system to ignore boot errors.
- The sample.exe also connects to multiple servers via https (more information in the network section)

3.2 File Monitoring

After deep analysis of the file system, I was able to see that the PE encrypted all the files on the system and changed it to e3sywy-readme.txt and this file includes a message from the threat actor on how to decrypt my data. From Procmon the operations where (1. create file, 2. Query directory 3. Close file)

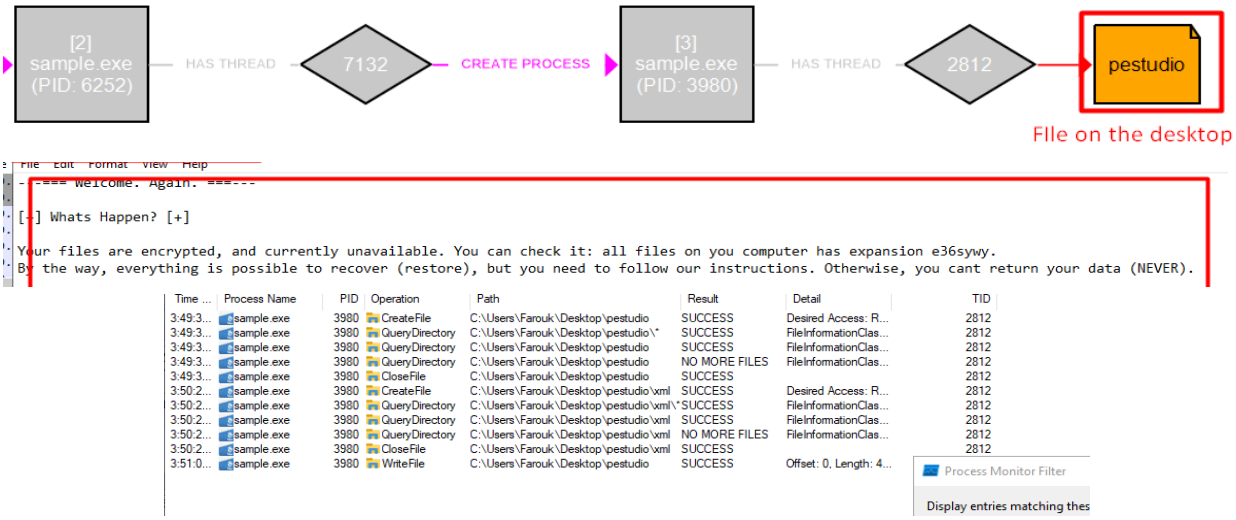


Figure 2 File Monitoring

3.3 Network traffic monitoring

For the network monitoring section, I filtered the results in Procmon to only display the network traffic and opened Wireshark for extra deep analysis.

Some of the IPs and domains the file connects to		
188.114.97.7	37.202.7.169	165.3.42.195
23.185.0.2	185.32.57.142	158.65.212.229
214.140.208.35.bc.googleusercontent.com	0.0.201.35.bc.googleusercontent.com	

The file connects to the above IPs/domains and creates an encrypted TCP tunnel using TLS and connects via https on port 443 and this makes it difficult to understand what this file is sending to the remote servers.

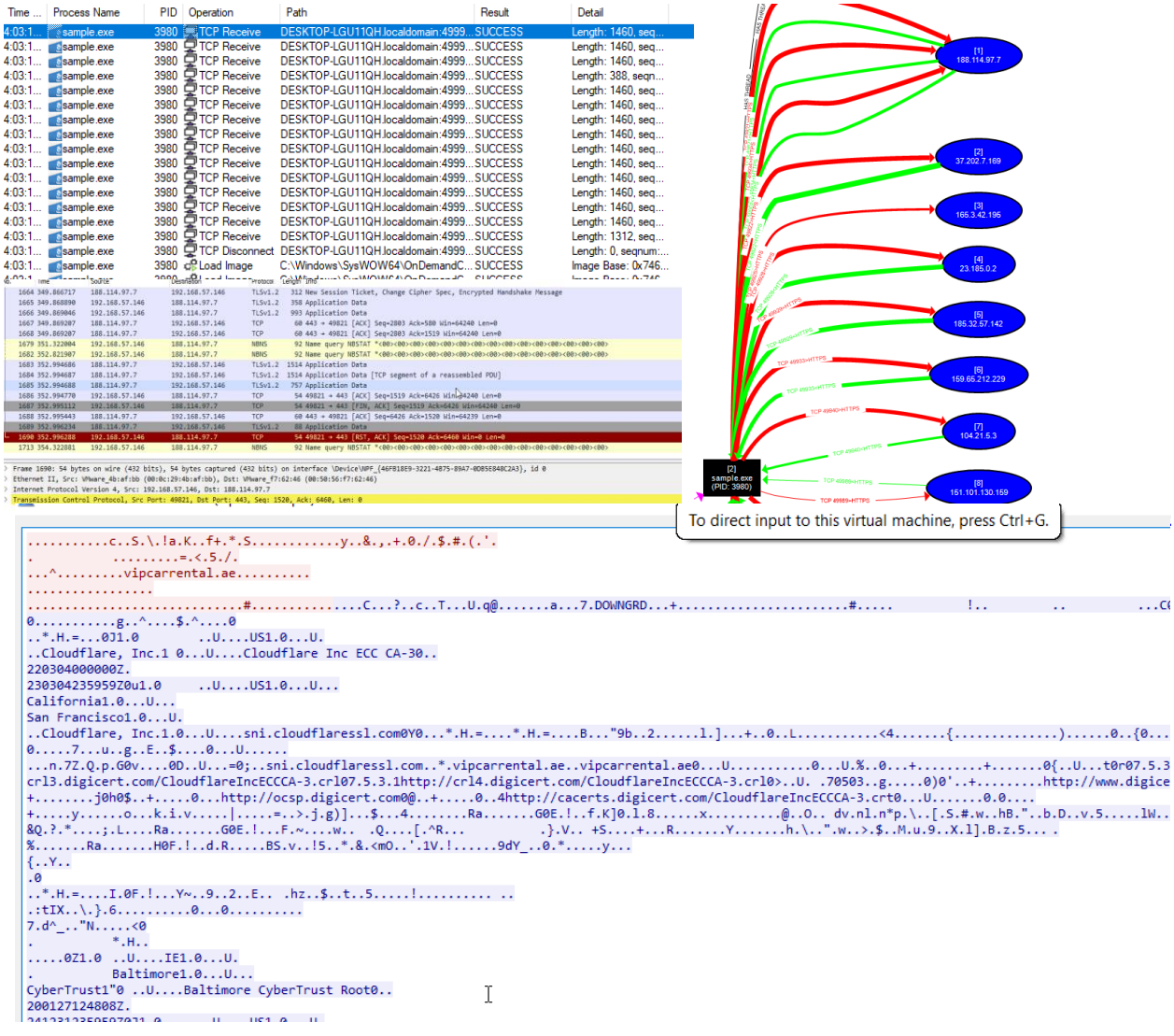


Figure 3 Network traffic monitoring screenshots

3.4 Extracting the Malware Config File

After knowing the malware family and conducting some research, I understood that the malware has a config file that is encrypted using RC4. This config file is in a separate section after unpacking the malware. In my instance the section name that holds the encrypted config file is named .s7bz. I used the CryptoTester tool by Michael Gillespie to brute force this section. The figure below is a part of the config file.


{"pk":"1g3/QEQPOQ7S3fBLZ0wvu/B9NfpLLvf8mByoN3or9E0=","pid":"5","sub":"367","dbg":false,"fast":true,
"wipe":true,"wht":{"fld":["windows","program files (x86)","$recycle.bin","programdata","boot",
"perflogs","appdata","mozilla","program files","intel","google","windows.old","tor browser",
"application data","system volume information","$windows.~ws","msocache","$windows.~bt"],"fls":
["ntuser.dat","boot.ini","autorun.inf","ntuser.ini","thumbs.db","ntldr","bootsect.bak","ntuser.dat.
log","iconcache.db","bootfont.bin","desktop.ini"],"ext":["icl","nomedia","msc","ldf","diagcab",
"drv","msp","key","wpx","idx","386","lock","rom","icns","msstyles","dll","hlp","sys","ics",
"diagcfg","shs","adv","ani","ocx","nls","scr","hta","bat","lnk","cpl","ico","spl","deskthemepack",
"bin","msu","themepack","mpa","msi","prf","rtp","com","ps1","theme","exe","cab","cmd","mod",
"diagpkg","cur"]},"wfld":["backup"],"prc":["wordpad.exe","outlook.exe","tbirdconfig.exe","agntsvc.
exe","thebat.exe","mydesktopservice.exe","sqbcoreservice.exe","thunderbird.exe","ocomm.exe","excel.
exe","thebat64.exe","steam.exe","xfssvccon.exe","firefoxconfig.exe","sqlagent.exe","ocssd.exe",
"mydesktopqos.exe","msaccess.exe","isqlplussvc.exe","mspub.exe","winword.exe","sqlbrowser.exe",
"dbeng50.exe","sqlservr.exe","oracle.exe","encsvc.exe","powerpnt.exe","dbsnmp.exe","infopath.exe",
"ocautoupds.exe","mysqld_opt.exe","visio.exe","msftesql.exe","mysqld_nt.exe","synctime.exe",
"sqlwriter.exe","mysqld.exe","onenote.exe"],"dmn":"craftingalegacy.com;g2mediainc.com;brinkdoepke.
eu;vipcarrental.ae;autoteamlast.de;hostastay.com;gavelmasters.com;ronaldhendriks.nl;successcolony.
com.ng;medicalsupportco.com;kompresory-opravy.com;sveneulberg.de;oththukaruva.com;voetbalhoogeveen.
nl;selected-minds.de;log-barn.co.uk;fsbforsale.com;jobkiwi.com.ng;ivancacu.com;11.in.ua;irizar.com;
colored-shelves.com;soundseeing.net;scotlandsroute66.co.uk;hawaiisteelbuilding.com;mindfuelers.com;

Figure 4 Extracting the Malware Config File

3.5 Summary of findings (Basic Dynamic Analysis)

The sample.exe file is a variant of the **Sodin ransomware family**, which is a malicious software designed to encrypt all files on the system and establish an encrypted TCP tunnel using TLS on port 443 to connect to multiple servers. The malware also disables recovery mode and sets the system to ignore boot errors, making it difficult to recover the encrypted files without paying the ransom. The malware also uses the NBNS protocol to propagate itself to other Windows devices on the network. The C2 server connection could not be determined as the malware employs a technique of sending fake DNS requests. Further analysis, such as reverse engineering, may be necessary to understand the encryption algorism and investigate the presence of any potential configuration files.

Section 4: Advanced Static Analysis

4.1 Function One (RC4 - Strings Encryption)

- **Function Names: sub_22489C [part 1], sub_224E03 [part 2], sub_2259FC [part 3]**

sub_22489C [part 1] is calling the function **sub_224E03[part 2]** and passing it a pointer address, the values 1982, 15, and 86, and a buffer named "Name" (which is an array of 44 WCHARs).

sub_224E03[part 2] then calls function **sub_2259FC [part 3]** and passes it the following 5 arguments:

- (pointer_address + 1982) - This is being used as the first part of the key for the RC4 algorithm.
- 15 - This is being used as the second part of the key for the RC4 algorithm.
- (pointer_address + 1982 + 15) - being used as the third part of the key for the RC4 algorithm.
- 86 - will be used later as the output length.
- Name - will be used as the input to encrypt.

sub_2259FC [part 3] function starts by initializing an array of 256 bytes called v14 and filling it with the values from 0 to 255. Then it uses the values of pointer_address and 1982 in a loop to shuffle the elements of the v14 array. This is known as the key-scheduling algorithm (KSA) of RC4.

After that, it enters another loop where it uses the shuffled v14 array to encrypt the byte array passed as input a5. It uses the values of 15 and 86 to control the number of iterations of this loop. The encryption process is done by XORing each byte of the input array a5 with a value from the v14 array, which is calculated using the current index of the loop and the values of the v14 array at the current and previous index. Finally, the function returns the encrypted byte array.

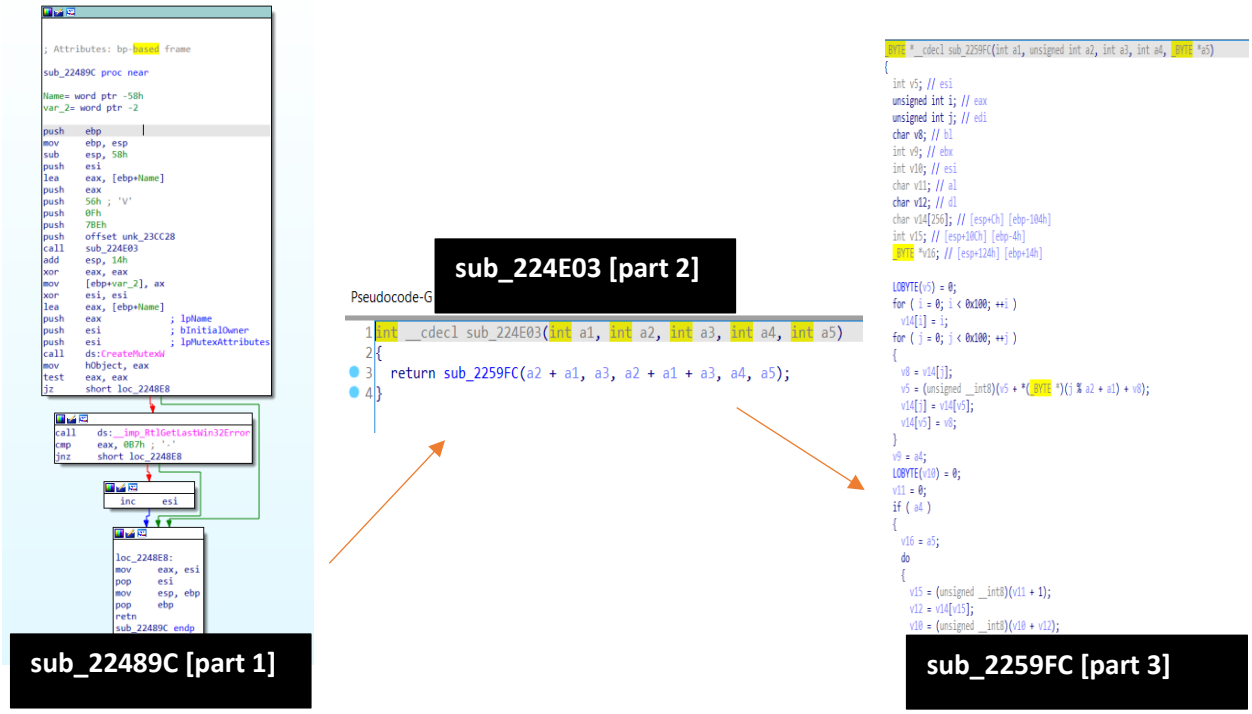


Figure 5 Function One (RC4 - Strings Encryption)

4.2 Function Two (listing Directories and Files)

- **Function Names: sub_2235FA [part 1], sub_ 2265E2 [part 2], sub_ [part 3]**

sub_2235FA [part 1] start by initiate variables of ones and zeros then calls the **sub_ 2265E2 [part 2]**

sub_ 2265E2 [part 2] appears to be a function that takes in one parameter hFindFile which is a handle to a file or directory. The function first calls sub_223C1E(0xFFFEu) which is likely to allocate memory and returns the pointer to the memory. It then enters a loop, where it checks if the value of the second byte of the pointer to the allocated memory is less than or equal to 0x5A(90) and inside the loop, it calls GetDriveTypeW function with the pointer to the allocated memory as the parameter. The function returns the drive type. If the drive type is less than or equal to 2, it calls **sub_226299[part 3]** function with the pointer to the allocated memory and hFindFile as the parameters.

sub_226299[part 3] function appears to start by calling a function (likely the hFindFile handle) passing the lpFileName and 0, and checks if it returned a non-zero value. If so, it calls another function passing lpFileName, and updates an internal variable (likely a counter) by the returned value. Then it enters a while loop that continues while the hFindFile handle returns 0. In the loop, it checks whether the internal variable is non-zero, and if so, it calls two functions to release memory and remove the current file name from the list. Then it calls a function that adds a "*" character to the end of the lpFileName and uses the FindFirstFileW function to search for files and directories in the current path.

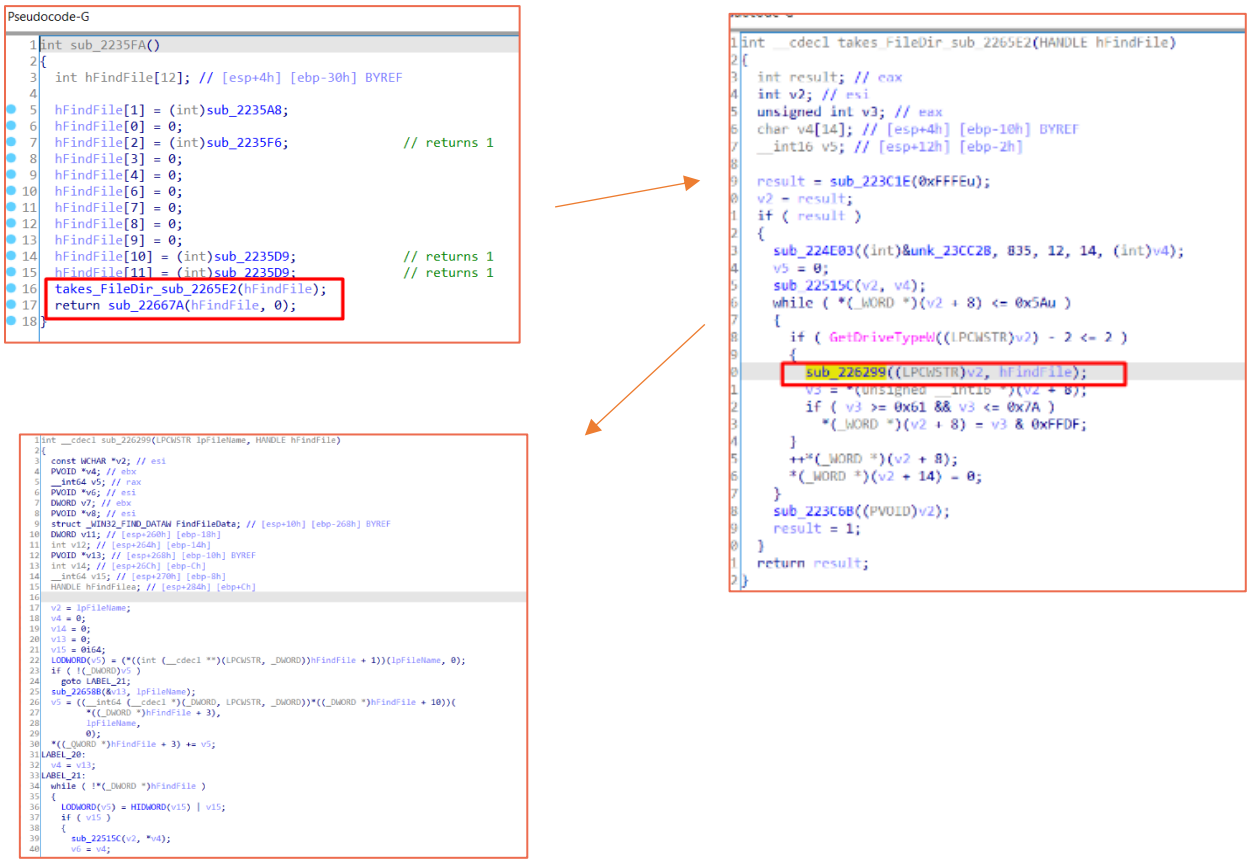


Figure 6 Function Two (listing Directories and Files)

4.3 Function Three (Salsa20 Encryption)

This ransomware employs the Salsa20 encryption algorithm to encrypt targeted files by using a pseudorandom key stream generated from the Salsa20 function. It appends a random extension with a length of 3 characters which is composed of alphanumeric characters to the original filenames

While preforming the advanced static analysis I found that most function in the program lead to one main function witch I named **sub_226E79[part 1]** and this function then calls 2 other functions **sub_226EF1[part 2]** and **sub_2259D7[part 3]**.

sub_226EF1[part 2] function starts by performing a series of bitwise operations on the elements of the array, which includes a combination of rotation, and bitwise AND operations. It then checks the value of the a3 variable, if it's equal to 128, the function performs some XOR operations on the elements of the array and updates them with new values. And then enters a loop which runs for 10 iterations. In each iteration, the function performs some operations on 4 elements of the array, including XOR and bitwise AND operations. It's most likely a function that is a part of a cryptographic algorithm, possibly related to key scheduling or key expansion.

sub_2259D7[part 3] function is a simple implementation of a stream cipher encryption/decryption function. It starts by initializing a variable v4 to the value of the third argument (a3) and a pointer v5 to the start of the array (a1). It then enters a loop which will run as many times as the value of v4. In each iteration of the loop, the function performs a bitwise XOR operation between the current byte pointed by v5 and the byte located at the position of v5 + (a2 - (_DWORD)a1). The function returns the original pointer to the start of the array (a1) after the loop is finished.

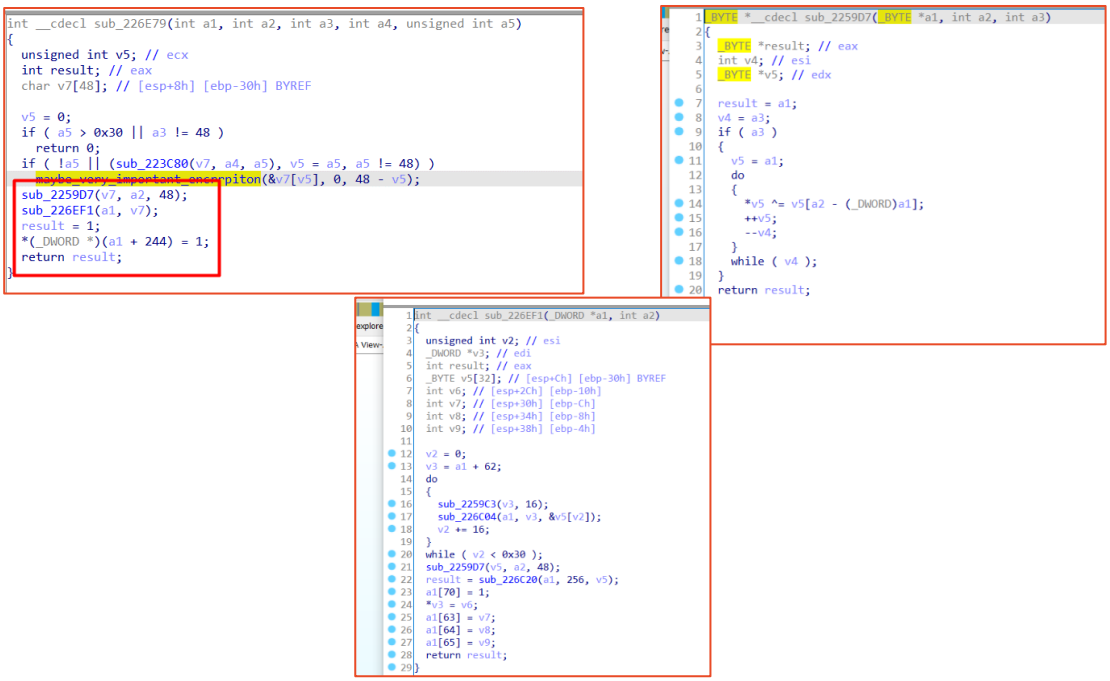


Table 7 Function Three (Salsa20 Encryption)

4.4 Function Four (Communication with C2 Servers)

- **Function Name: sub_226826**

sub_226826 appears to be a function that makes an HTTP request to a specified URL. The function takes four parameters: a pointer to a wide string representing the URL (pwszUrl), a pointer to optional data to be sent with the request (lpOptional), the length of the optional data (dwOptionalLength), and two integers (a4 and a5) which it is not clear what they are used for.

The function starts by initializing some variables, including an array of characters (pszAgentW) with a specific string and a structure (UrlComponents) used to hold information about the URL.

Then it opens an internet session using the WinHttpOpen function and uses the WinHttpCrackUrl function to parse the URL and fill the UrlComponents structure with the information. Then it connects to the host using the WinHttpConnect function and opens an HTTP request using the WinHttpOpenRequest function. It is likely that the function is used to send HTTP requests and retrieve the response from a server.

```
sub_224E03(&unk_23CC28, 1255, 4, 144, pszAgentW);
v5 = 0;
pszAgentW[72] = 0;
v6 = WinHttpOpen(pszAgentW, 0, 0, 0, 0);
result = 0;
dwBufferLength = (DWORD)v6;
if ( v6 )
{
  UrlComponents.nPort = 0;
  UrlComponents.dwStructSize = 60;
  UrlComponents.lpszScheme = 0;
  UrlComponents.dwSchemeLength = 0;
  UrlComponents.nScheme = INTERNET_SCHEME_DEFAULT;
  UrlComponents.lpszHostName = 0;
  UrlComponents.dwHostNameLength = 1;
  UrlComponents.lpszUserName = 0;
  UrlComponents.dwUserNameLength = 0;
  UrlComponents.lpszPassword = 0;
  UrlComponents.dwPasswordLength = 0;
  UrlComponents.lpszUrlPath = 0;
  UrlComponents.dwUrlPathLength = 1;
  UrlComponents.lpszExtraInfo = 0;
  UrlComponents.dwExtraInfoLength = 0;
  if ( !WinHttpCrackUrl(pwszUrl, 0, 0, &UrlComponents) )
  {
    WinHttpCloseHandle(v6);
    return 0;
  }
  *(_WORD *)&UrlComponents.lpszHostName[2 * UrlComponents.dwHostNameLength] = 0;
  v8 = WinHttpConnect(v6, (LPCWSTR)UrlComponents.lpszHostName, UrlComponents.nPort, 0);
  hInternet = v8;
  if ( !v8 )
  {
    WinHttpCloseHandle(v6);
    return 0;
  }
  if ( !*(_WORD *)UrlComponents.lpszUrlPath )
    *(_WORD *)UrlComponents.lpszUrlPath = 47;
  sub_224E03(&unk_23CC28, 1767, 15, 8, pwszVerb);
  v9 = 0;
```

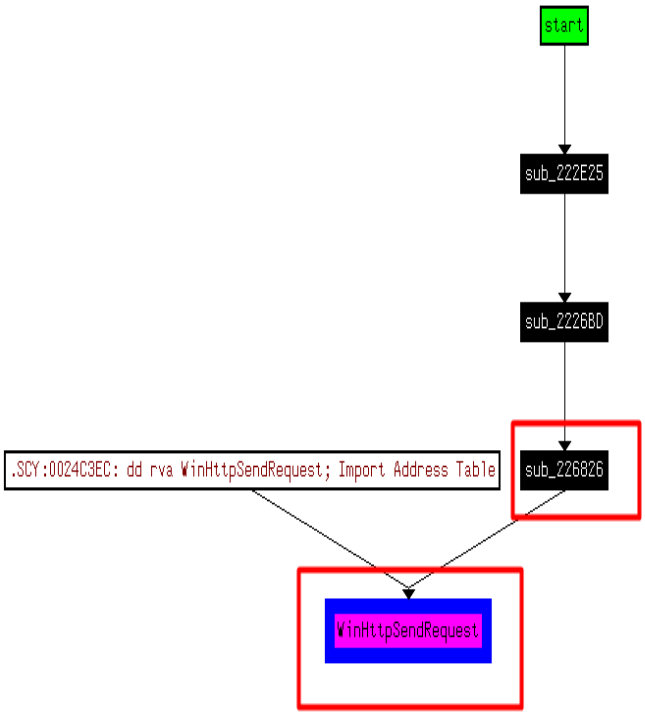
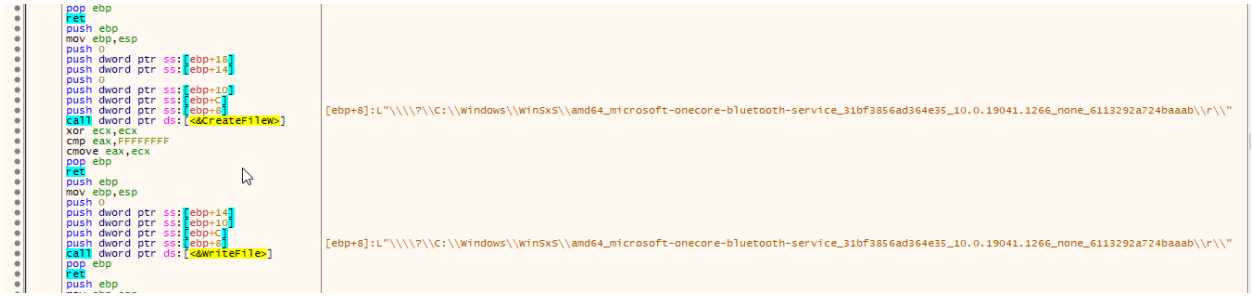


Figure 7 Function Three (Communication with C2 Servers)

Section 5: Advanced Dynamic Analysis

For this section I used x32 debugger to debug the sample PE and try to prove that the 4 functions mentioned above in the [Advanced Static Analysis] preform the operations I concluded above.



Section 6: Conclusion

The following table presents a comprehensive summary of the key properties and operational characteristics exhibited by the malware sample, along with the corresponding sections in the report where further details and analysis can be found.

Basic Properties/Behavior	Details
The malware is Packed	The malware is custom packed and had to be manually unpacked. More details were discussed in the Unpacking the Malware sample section of this report.
The malware uses a Json config file	The configuration used is an encrypted Json file. More details in the Extracting the Malware Config File section of the report.
The malware encrypts all the data on the device	This ransomware employs the Salsa20 encryption algorithm to encrypt the targeted files. The encrypted files have their original filenames modified by appending a random-generated alphanumeric extension. More details are in the Function Three (Salsa20 Encryption) section of the report.
The malware deletes backups and shadow copies	The malware implements a routine to eradicate shadow copies of both individual files and system volumes, effectively hindering any attempts at recovery for the encrypted files. More details are explained in the Process Monitoring section of this report
The malware communicates with multiple C2 servers	After the encryption process the malware generates randomized URLs for specific domains specified in its configuration file and sends the keys to the C2 servers. More details are explained in the Function Four (Communication with C2 Servers) section and the Network traffic monitoring section of the report.

Table 8 Conclusion

Appendices

Yara Rules for Detecting This Malware

```
rule CW2_Revil{

    meta:

        last_updated = "2023-1-15"

        author = "Farouk"

        description = "Revil Ransomware"

    strings:

        $string1 = "duyumavohevifafajesibicepuxidu maxebocugazimehuxadixi citulosizinivuxolifiri vin" ascii

        $string2 = " jilidimizicanorukepu nabufoxipaxo" ascii

        $PE_magic_byte = "MZ"

        $suspicious_hex_string = {66 C7 05 F3 AE  46 00 6C 00}

    condition:

        $PE_magic_byte at 0 and

        ($string1 and $string2) or

        $ suspicious_hex_string

}
```