Reverse Engineering Project

Naif Alkaltham / Cody Blanchard

Malware: LuckyCat.exe

SOURCE:
https://tria.ge/220124-dge1wsagc5

MD5

9f9723c5ff4ec1b7f08eb2005632b8b1

SHA-1

e47a821ef85d722f01f10adff227f45552e4ec73

SHA-256

e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4

Introduction

LuckyCat is a dangerous malware that is commonly used in cyber attacks. It is often distributed via phishing emails or links that trick victims into downloading and installing the malware on their computers. Once installed, LuckyCat establishes a connection to a remote server operated by the attackers.

LuckyCat's ability to remain undetected by most antivirus software is particularly alarming. The malware employs advanced techniques to conceal its presence on the infected system, putting sensitive data at risk. Our reverse engineering project aims to analyze the LuckyCat for Windows malware using a variety of static and dynamic analysis techniques, including disassembly, debugging, and memory forensics. The insights gained through this process will be used to develop effective detection and mitigation strategies that will help safeguard against future LuckyCat attacks.

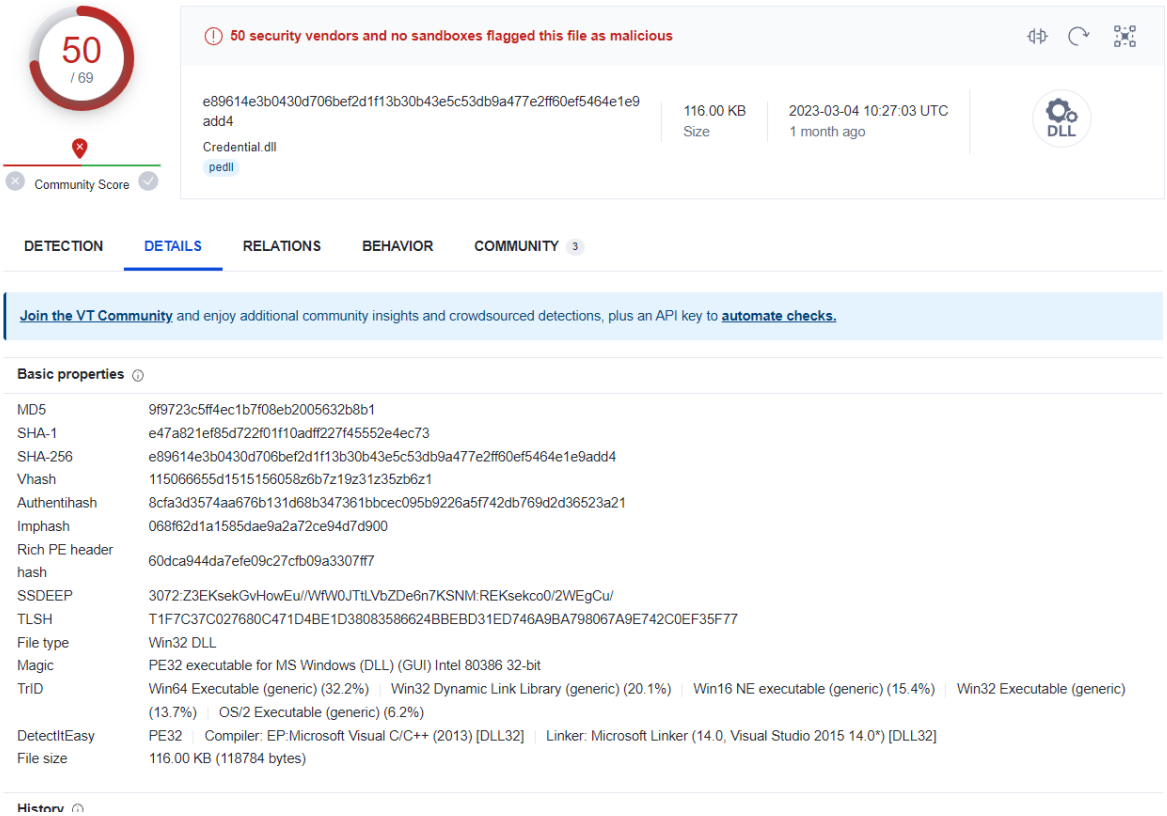
Analysis Stages:

1- Basic Static Analysis
2- Basic Dynamic Analysis:
3- Advanced Static Analysis:
4- Advanced Dynamic Analysis:

Basic Static Analysis:

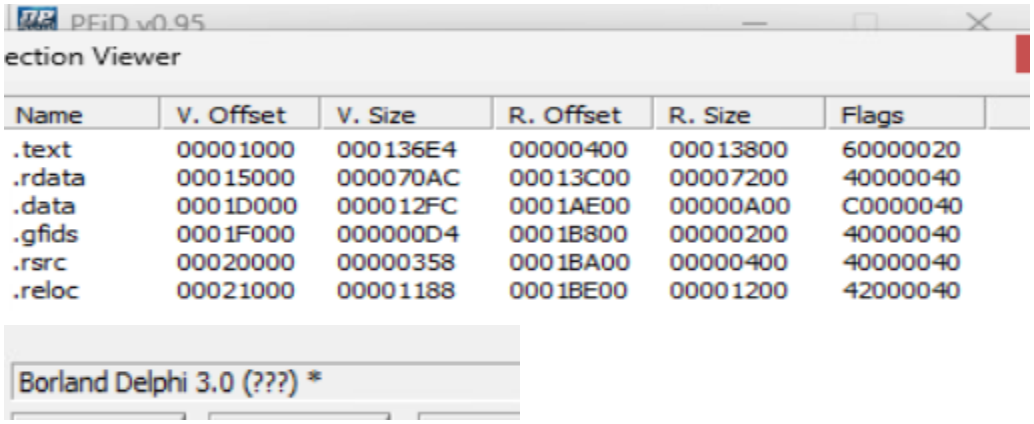
1- Virustotal

https://www.virustotal.com/gui/file/e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4/details



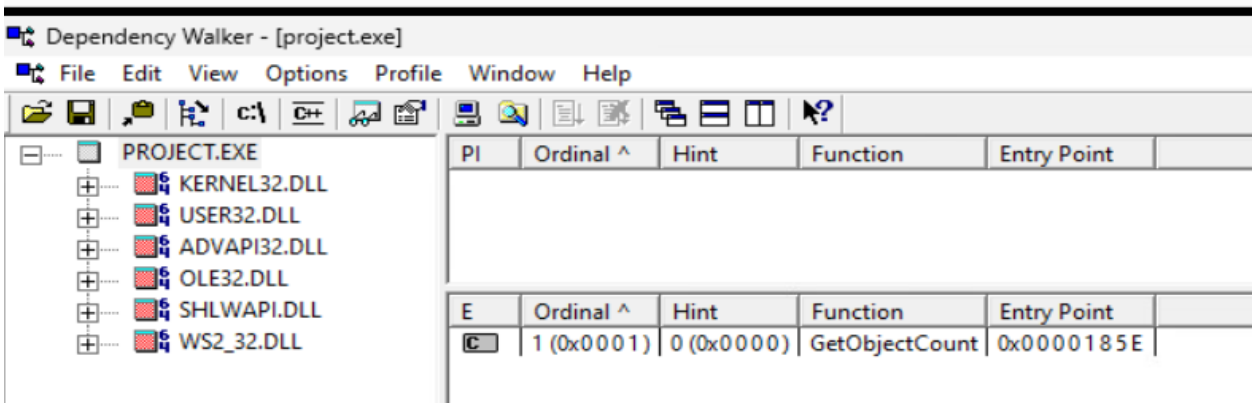
2- PEiD

The malware, identified as Borland Delphi 3, appears to be utilizing a technique called "process replacement" to conceal itself on the system by replacing a process's memory with its own code, as evidenced by its larger virtual size compared to its row size.

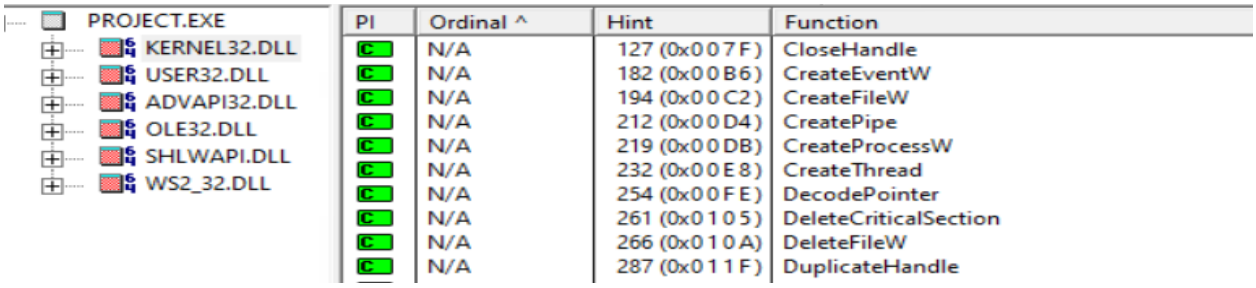


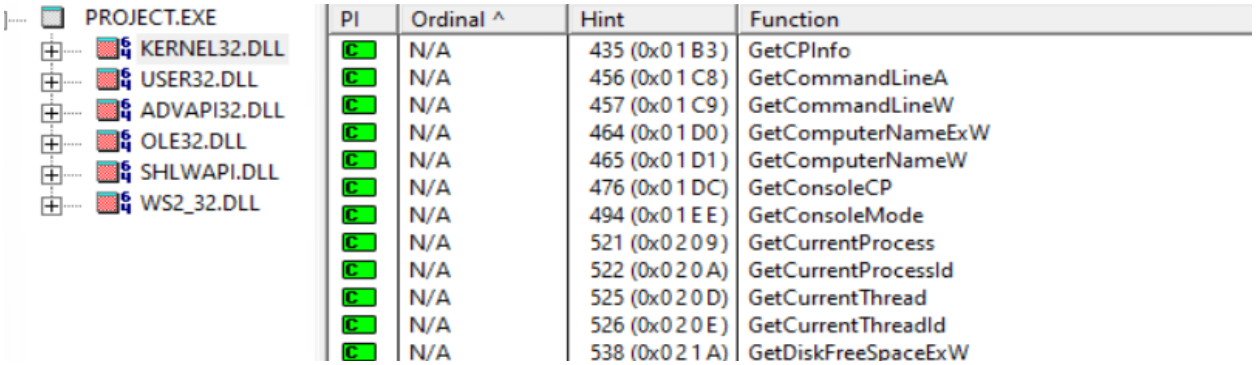
3- Dependency Walker

First thing that I noticed using dependency walker that there are 6 main DLLs is used by the malware which are KERNEL32.DLL, USER32.DLL, ADVAPI32.DLL,OLE32.DLL,SHLWAPI.DLL,W2_32.DLL that are begging called by function called GetObjectCount.

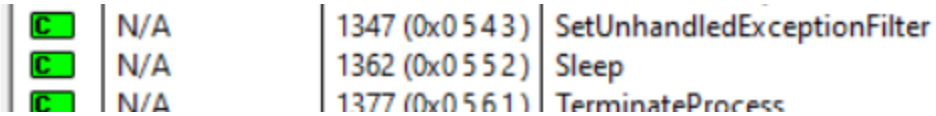


Second, I saw that there are some functions in the KERNEL.DLL that show a sign of file manipulation such as creating a file and deleting a file and also there are functions that will collect information about the victim's computer.





Lastly, we found that there is a sleep function which is commonly used to hide the malware due the fact that it can help the malware to not be detected by the anti-malware tools.



4- Strings

First, we found that there are a lot of sub-dlls that will allow malware to function and some more DLLs that we didn't see in Dependency Walker that will be executed when running the malware.

```
api-ms-win-appmodel-runtime-l1-1-1
api-ms-win-core-datetime-l1-1-1
api-ms-win-core-file-l2-1-1
api-ms-win-core-localization-l1-2-1
api-ms-win-core-localization-obsolete-l1-2-0
api-ms-win-core-processthreads-l1-1-2
api-ms-win-core-string-l1-1-0
api-ms-win-core-sysinfo-l1-2-1
api-ms-win-core-winrt-l1-1-0
api-ms-win-core-xstate-l2-1-0
api-ms-win-rtcore-ntuser-window-l1-1-0
api-ms-win-security-systemfunctions-l1-1-0
ext-ms-win-kernel32-package-current-l1-1-0
ext-ms-win-ntuser-dialogbox-l1-1-0
ext-ms-win-ntuser-windowstation-l1-1-0
```

```
ReadFile
WriteFile
DeleteFileW
GetTempPathW
GetCurrentThreadId
GetDiskFreeSpaceExW
GetVolumeInformationW
```

Organizing the screenshots

```
Credential.dll
```

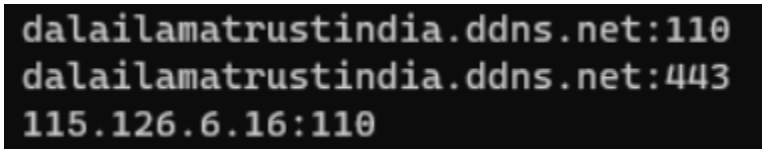
```
ADVAPI32.dll
```

```
mscoree.dll
```

Second, in this picture, we noticed that there might be a connection to a web-service. Which might be a connection to the hacker server to either download or upload files.

```
DLL---Start connect to %ws:%d
x64
```

Also, here are some examples that we think the malware will connect to by ports 110 and 443 which are for the POP3 protocol that is for unencrypted access to email and https connection.



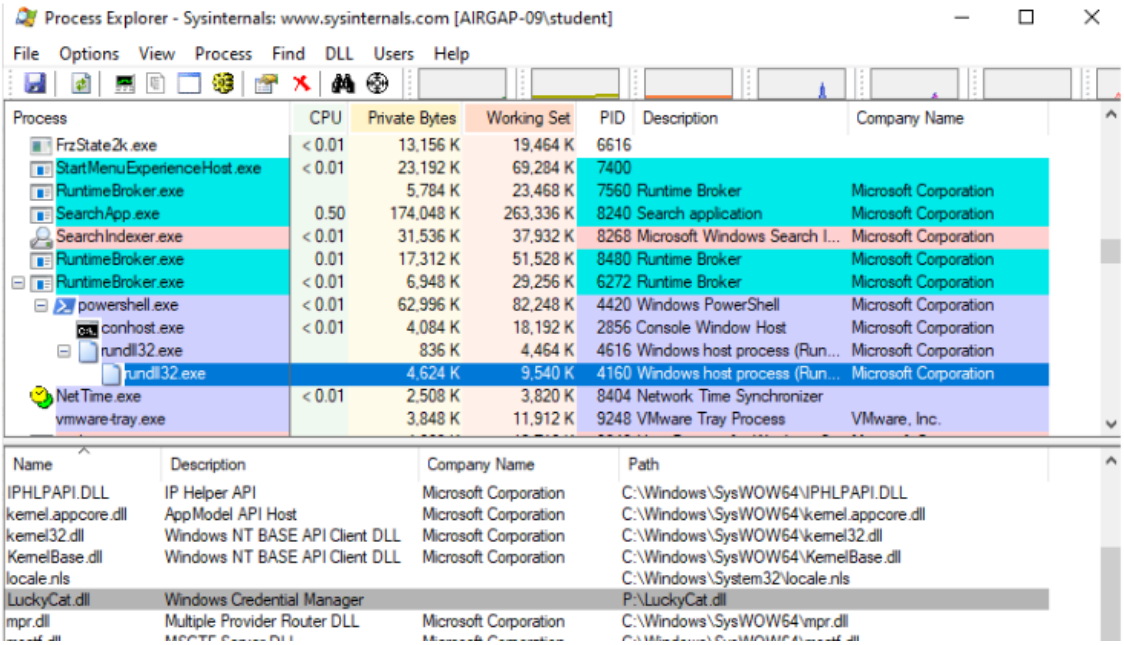
Basic Dynamic Analysis:

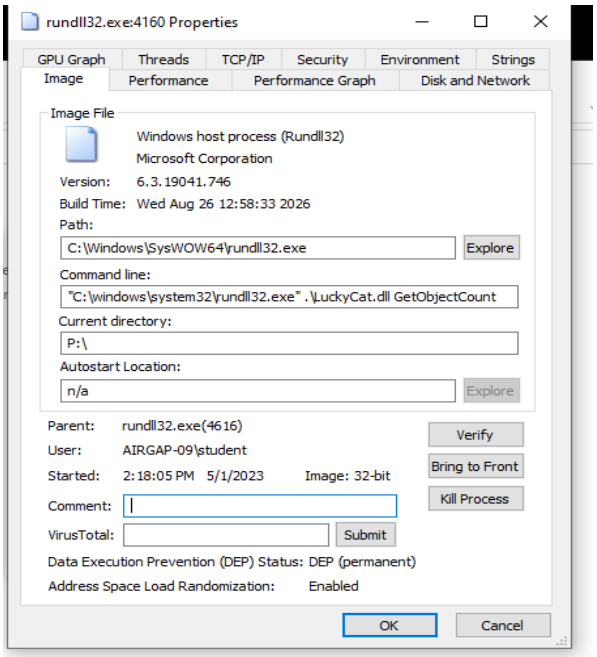
In this section, our main goal is to confirm that the malware is running through different tools. Also, to compare and investigate any possible information that would relate to the information we got from the Basic Static Analysis.

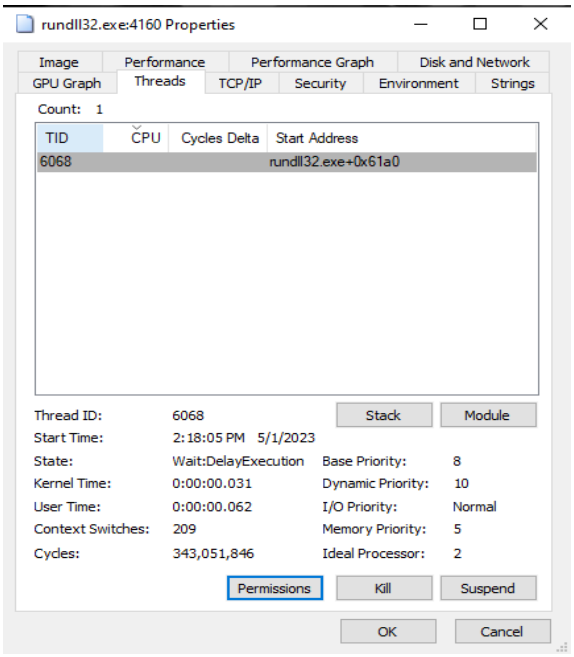
First, we ran the 32-bit dynamic-link library with rundll32.exe.

```
PS P:\> rundll32.exe .\LuckyCat.dll GetObjectCount
PS P:\>
```

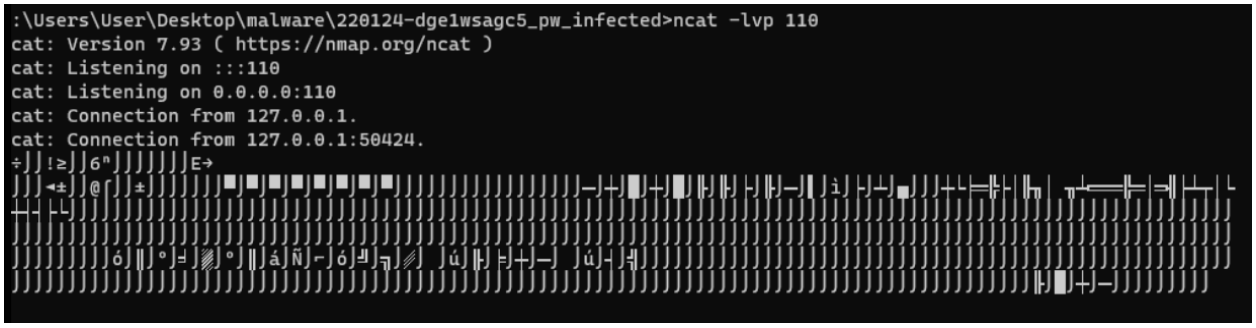
Next, we opened Process Explorer and found that indeed LuckyCat.dll was running.

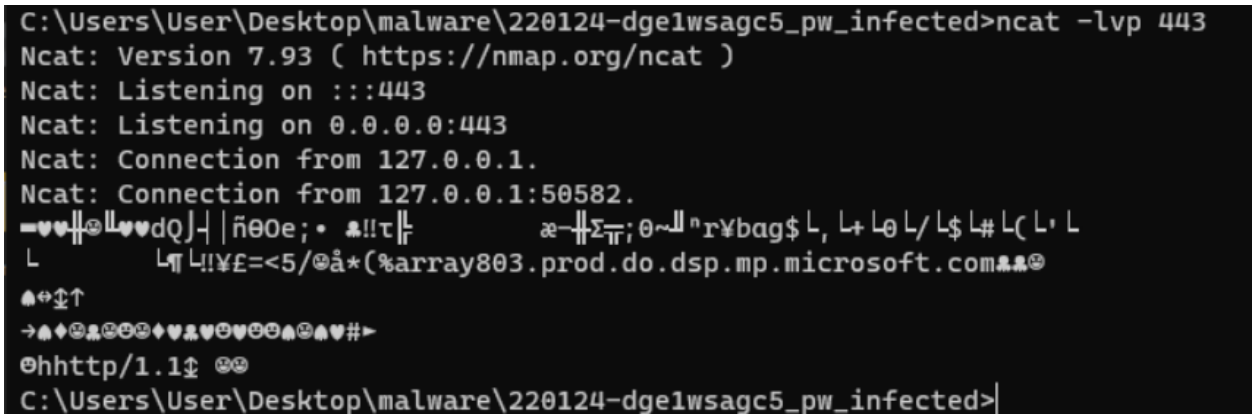






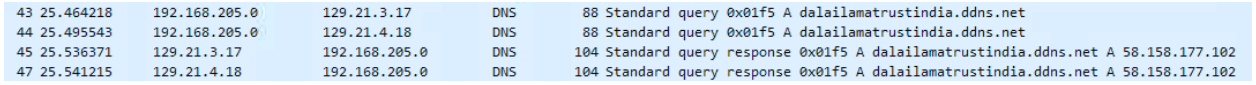
Second, we wanted to confirm that there will be connections for to the ports we found in the Basic Static Analysis, and here are some connections that have been captured by the Ncat and the Apatedns.





Time	Domain Requested	DNS Returned
22:37:15	dalailamatrustindia.ddns.net	FOUND
22:37:23	dalailamatrustindia.ddns.net	FOUND
22:37:23	dalailamatrustindia.ddns.net	FOUND
22:37:34	dalailamatrustindia.ddns.net	FOUND
22:37:34	ade.googlesyndication.com	FOUND
22:37:34	ade.googlesyndication.com	FOUND
22:37:49	dalailamatrustindia.ddns.net	FOUND
22:37:55	dalailamatrustindia.ddns.net	FOUND

Also, we had started capturing network traffic with Wireshark before starting the malware and LuckyCat.dll appears to be making DNS queries. The malware seems to be attempting to find the address of dalailamatrustindia.ddns.net as seen in the Wireshark capture screenshot below.



This DNS traffic is consistent with one of the DNS resolutions VirusTotal says the malware makes to dalailamatrustindia.ddns.net. Also, it states that LuckyCat.dll connects to 58.158.177.102 using ports 110 and 443

DNS Resolutions

- 102.177.158.58.in-addr.arpa
- 208.118.189.20.in-addr.arpa
- 212.143.182.52.in-addr.arpa
- 29.73.42.20.in-addr.arpa
- 6.160.190.20.in-addr.arpa
- 9.31.126.40.in-addr.arpa
- 94.16.208.104.in-addr.arpa
- dalailamatrustindia.ddns.net

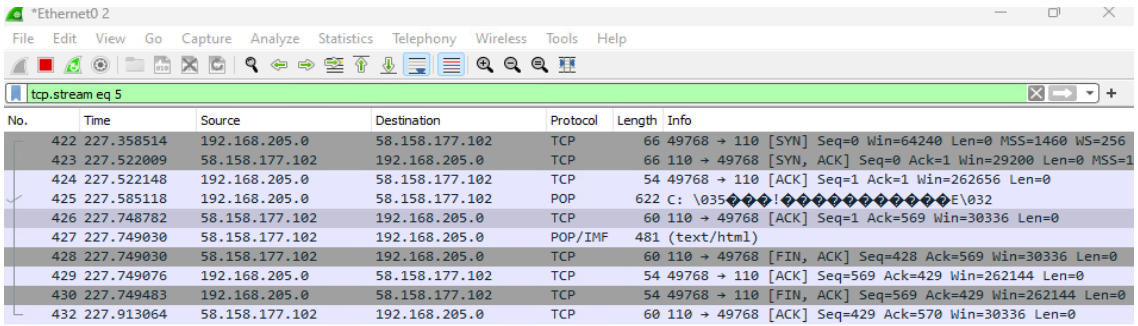
IP Traffic

58.158.177.102:110 (TCP)
58.158.177.102:443 (TCP)

Promon shows LuckyCat attempting to start a TCP connection

2:34:4...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1678 -> AIRGAP-09:pop3	SUCCESS	Length: 0, seqnum:...
2:34:4...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1678 -> AIRGAP-09:pop3	SUCCESS	Length: 0, seqnum:...
2:34:4...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1678 -> AIRGAP-09:pop3	SUCCESS	Length: 0, seqnum:...
2:34:4...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1678 -> AIRGAP-09:pop3	SUCCESS	Length: 0, seqnum:...
2:34:4...	rundll32.exe	4160	TCP Disconnect	AIRGAP-09:1678 -> AIRGAP-09:pop3	SUCCESS	Length: 0, seqnum:...
2:35:5...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1703 -> AIRGAP-09:https	SUCCESS	Length: 0, seqnum:...
2:35:5...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1703 -> AIRGAP-09:https	SUCCESS	Length: 0, seqnum:...
2:35:5...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1703 -> AIRGAP-09:https	SUCCESS	Length: 0, seqnum:...
2:35:5...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1703 -> AIRGAP-09:https	SUCCESS	Length: 0, seqnum:...
2:35:5...	rundll32.exe	4160	TCP Disconnect	AIRGAP-09:1703 -> AIRGAP-09:https	SUCCESS	Length: 0, seqnum:...
2:37:1...	rundll32.exe	4160	TCP Reconnect	AIRGAP-09:1739 -> AIRGAP-09:pop3	SUCCESS	Length: 0, seqnum:...

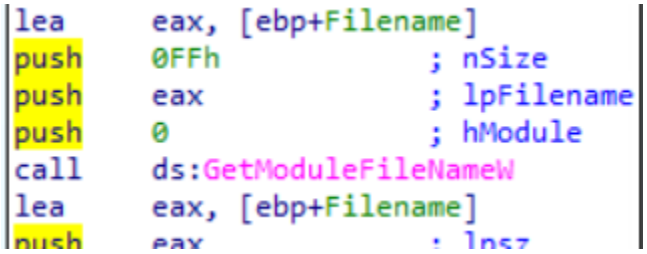
In our Wireshark capture we saw that LuckyCat.dll is sending TCP traffic to 58.158.177.102 on port 110.



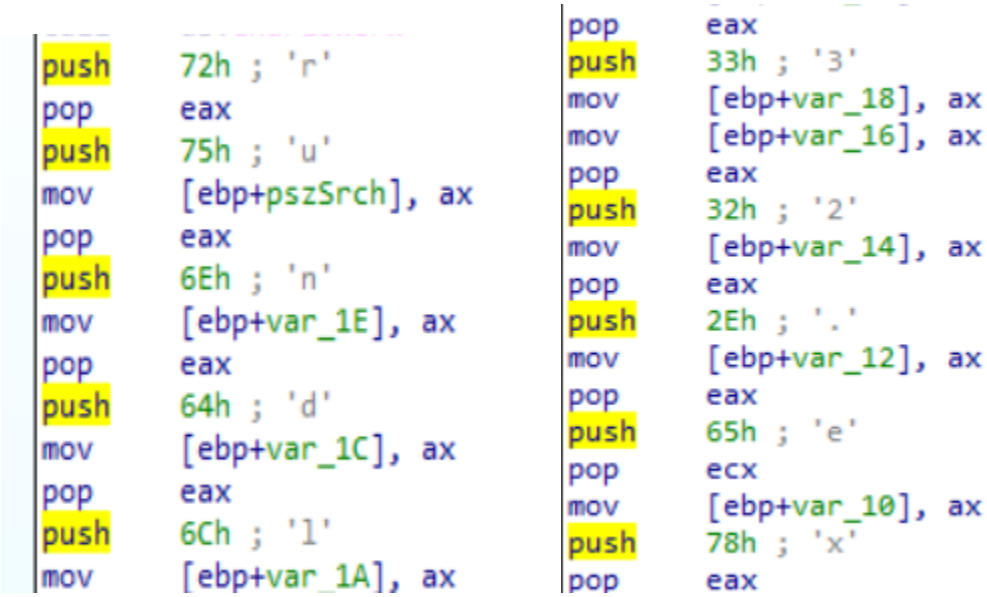
Advanced Static Analysis:

1- IDA

The first thing that we noticed is that there are three parameters that are being pushed to a function called (GetModuleFileNameW).



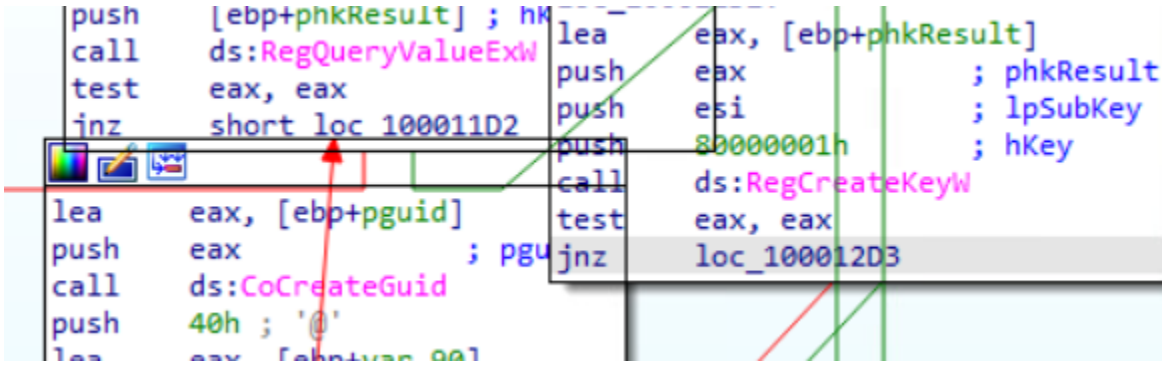
Also, we noticed that there is a string that has been spelled out in the malware which is (rundl32.ex) that we believe it would refer to (rundll32.exe) which is used to execute any type of dll.



Thus, we think that the code is checking whether the (rundll32.exe) is being used to run the malware's dll. Moreover, we saw another string that has been spelled out and copied character by character and we think this method is used to avoid the detection of copying the whole string at one time. Also, we saw a function called (RegOpenKeyW) which can be used as a placeholder for a registry key.

```
push    44h ; 'D'
pop     eax
push    65h ; 'e'
mov     [ebp+ValueName], ax
pop     eax
push    66h ; 'f'
mov     [ebp+var_1E], ax
pop     eax
push    61h ; 'a'
mov     [ebp+var_1C], ax
pop     eax
push    75h ; 'u'
mov     [ebp+var_1A], ax
pop     eax
push    6Ch ; 'l'
mov     [ebp+var_18], ax
push    74h ; 't'
pop     ecx
push    53h ; 'S'
mov     [ebp+var_16], ax
pop     eax
push    72h ; 'r'
mov     [ebp+var_12], ax
pop     eax
push    69h ; 'i'
mov     [ebp+var_E], ax
pop     eax
push    6Eh ; 'n'
mov     [ebp+var_C], ax
pop     eax
push    67h ; 'g'
push    80000001h       ; hKey
mov     [ebp+var_14], cx
mov     [ebp+var_10], cx
call    ds:RegOpenKeyW
test    eax, eax
jnz     loc_100012D3
```

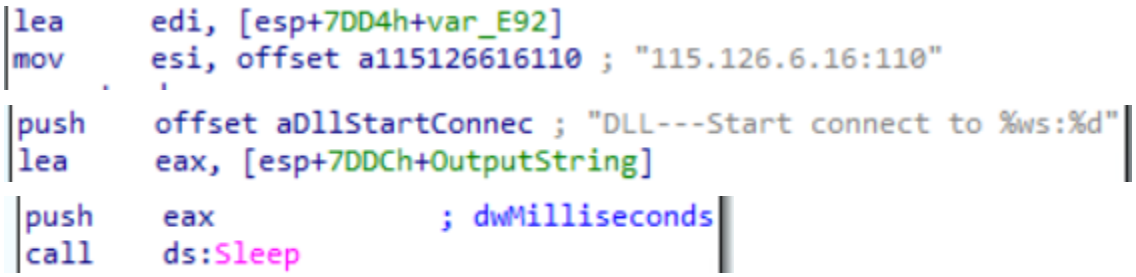
Also, we saw a couple more important functions that has been called especially (CoCreateGuid) which can be used to create a globally unique identifier that can be easily accessible and to be used to re-reference the same object again.



Lastly, here are some proofs that show the malware will try to attempt to connect to URLs and try to hide the malware using the sleep function.

```
mov     esi, offset aDalailamatrust ; "dalailamatrustindia.ddns.net:110"
lea     edi, [esp+7DD0h+var_F38]
```

```
mov     esi, offset aDalailamatrust_0 ; "dalailamatrustindia.ddns.net:443"
stosw
```

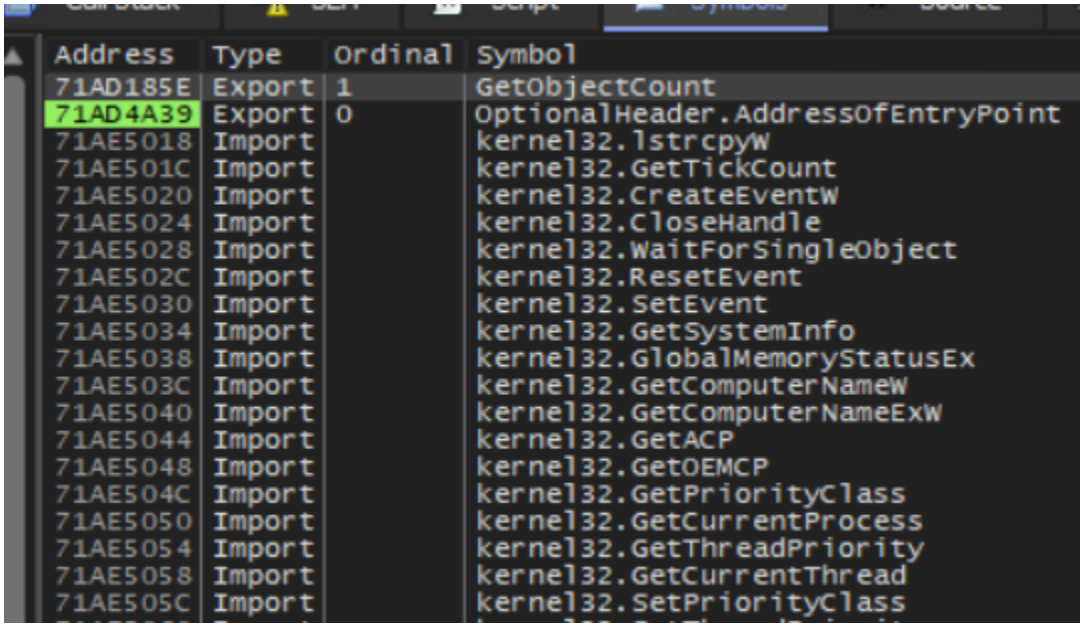


Therefore, we think that the malware is confirmed to have functions that do registry manipulation functions, and offsets with URLs to either download or upload files. Also, it has defensive measures that can be used to hide within the victim's computer.

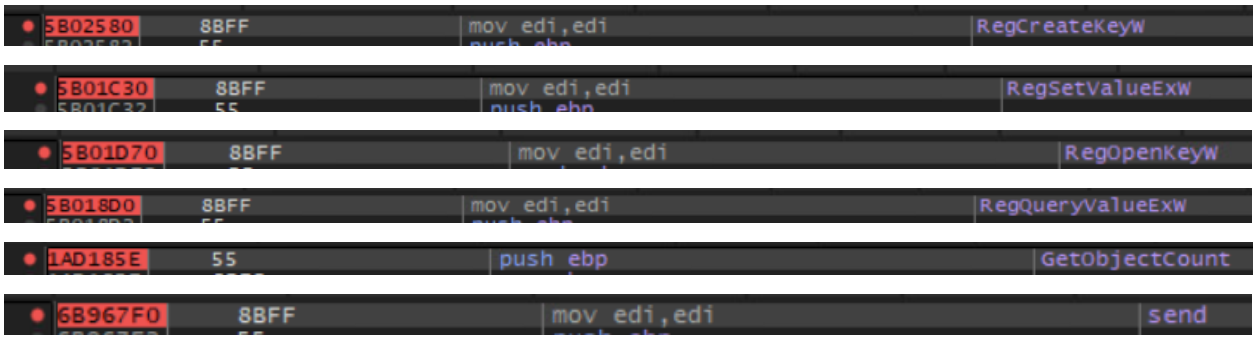
Advanced Dynamic Analysis:

1- x32dbg

In this section we started to analyze the export and import function, here are some examples. Also, we created a breakpoint for each of the registry functions that we think are important to analysis them.

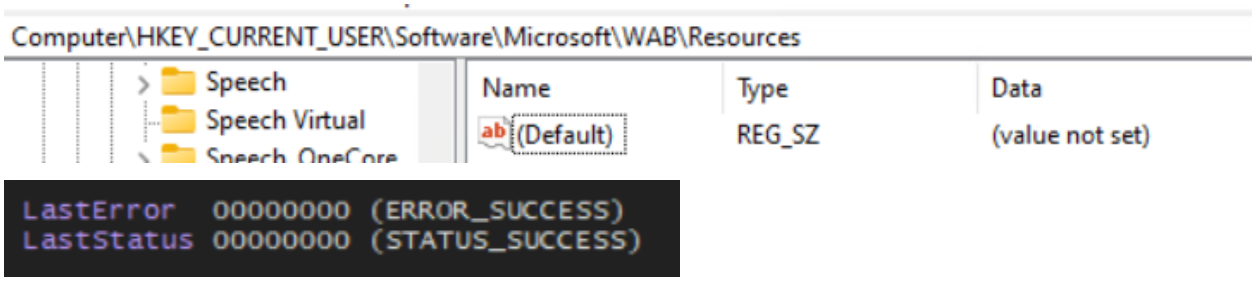


Address	Type	Ordinal	Symbol
71AD185E	Export	1	GetObjectCount
71AD4A39	Export	0	OptionalHeader.AddressOfEntryPoint
71AE5018	Import		kernel32.lstrcpyW
71AE501C	Import		kernel32.GetTickCount
71AE5020	Import		kernel32.CreateEventW
71AE5024	Import		kernel32.CloseHandle
71AE5028	Import		kernel32.WaitForSingleObject
71AE502C	Import		kernel32.ResetEvent
71AE5030	Import		kernel32.SetEvent
71AE5034	Import		kernel32.GetSystemInfo
71AE5038	Import		kernel32.GlobalMemoryStatusEx
71AE503C	Import		kernel32.GetComputerNameW
71AE5040	Import		kernel32.GetComputerNameExW
71AE5044	Import		kernel32.GetACP
71AE5048	Import		kernel32.GetOEMCP
71AE504C	Import		kernel32.GetPriorityClass
71AE5050	Import		kernel32.GetCurrentProcess
71AE5054	Import		kernel32.GetThreadPriority
71AE5058	Import		kernel32.GetCurrentThread
71AE505C	Import		kernel32.SetPriorityClass



```
5B02580   8BFF   mov edi,edi   RegCreateKeyW
5B01C30   8BFF   mov edi,edi   RegSetValueExW
5B01D70   8BFF   mov edi,edi   RegOpenKeyW
5B018D0   8BFF   mov edi,edi   RegQueryValueExW
1AD185E   55     push ebp      GetObjectCount
6B967F0   8BFF   mov edi,edi   send
```

Therefore, to test these functions we created a key registry so that we can notice and follow the malware, especially by the ReqOpenKey breakpoint, which was successful.



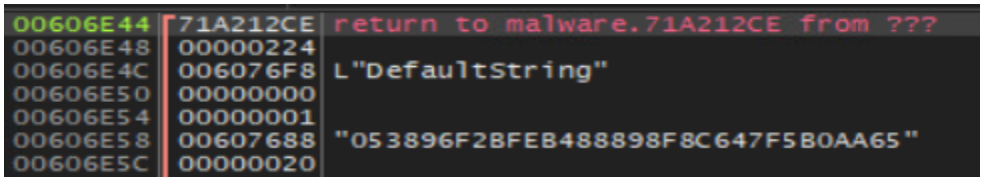
Computer\HKEY_CURRENT_USER\Software\Microsoft\WAB\Resources

Speech
Speech Virtual
Speech OneCore

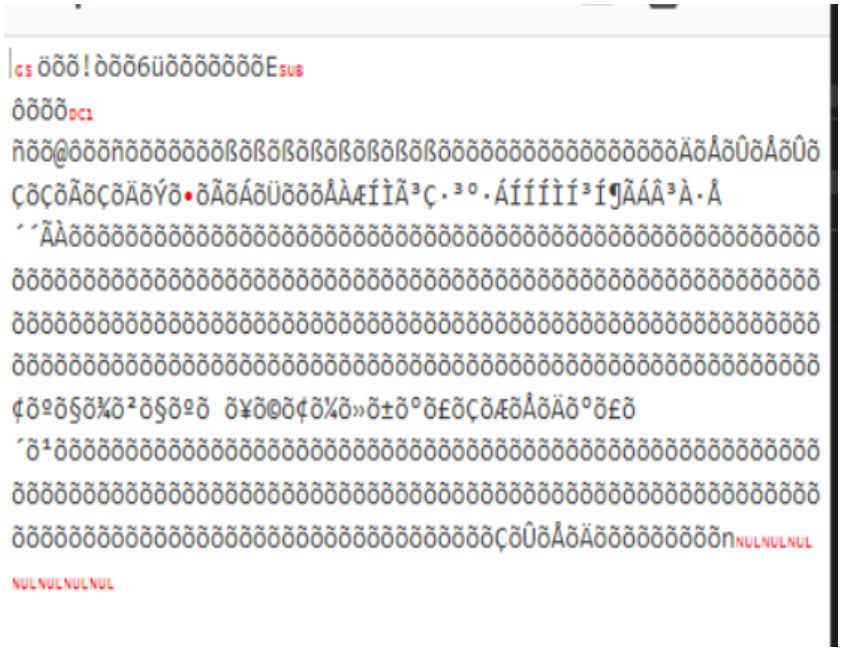
Name	Type	Data
(Default)	REG_SZ	(value not set)

```
LastError  00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)
```

In addition, here we can see that the string “DefaultString” that we saw before has obtained important information that we will analyze in the next steps.

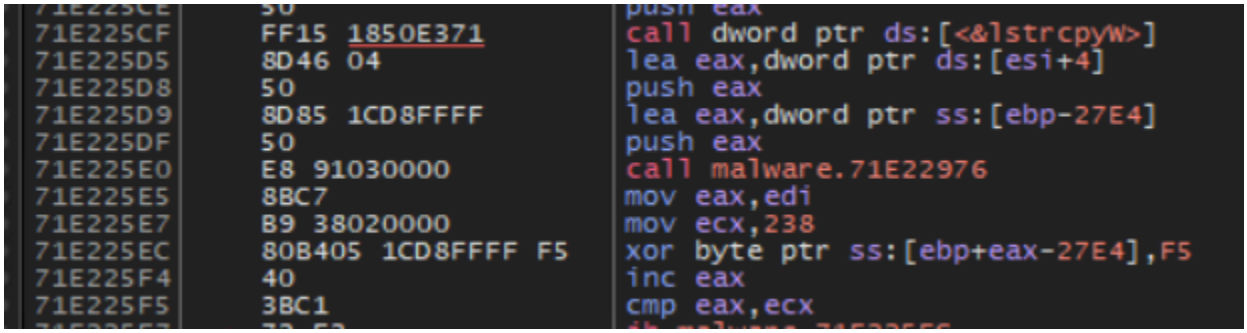


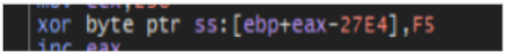
The first thing we decided is to extract the information that we got after following the edi registry from the send in the dump and analyze it using the cyberchef website with the HEXdump operation.

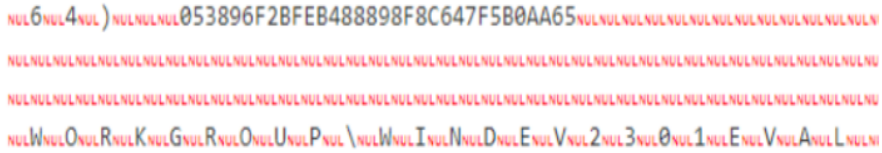


Next, we ran the malware in x32dbg and used the symbols page to find function calls. We discovered that x32dbg flagged one of these functions as a send function. Thinking that this function might have to do with networking we added a breakpoint and used the x32dbg step command to run the asm instructions after this breakpoint. Lower down in the binary we discovered that under the send breakpoint that the malware was using an xor operation with F5 to encrypt the information. We decided to decrypt the information that we found by using the xor operation and use F5 as a key Value. This gave us very important information that one of the things that has been sent from the malware was

the victim's computer name, which is **WinDev2301Eval**.

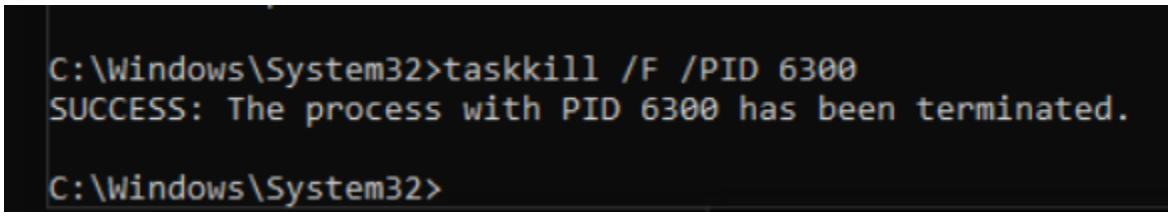


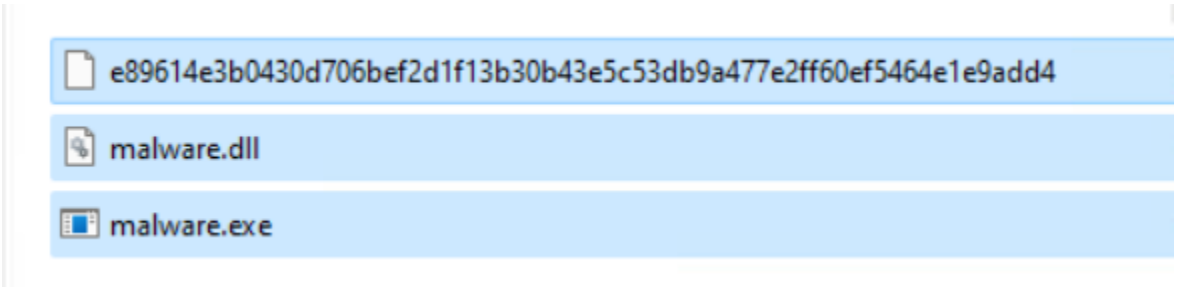




How to remove the malware

The first thing to do is to kill any running process under the malware name such as process number “6300” that we found using Process Explorer, and after that is to delete any files that are related to the malware.





Here is an example of how to find the process number using Process Explorer.

SUMMARY

The analysis of the LuckyCat malware involved four stages, namely Basic Static Analysis, Basic Dynamic Analysis, Advanced Static Analysis, and Advanced Dynamic Analysis. In the Basic Static Analysis stage, the malware was subjected to various tools, including PEiD and Strings analysis, to gain insights into its behavior. It was found that the malware employed process replacement to conceal itself and was using six main DLLs, including KERNEL32.DLL, USER32.DLL, and OLE32.DLL. A sleep function was also found, and a connection to a web service was detected.

The Basic Dynamic Analysis stage involved confirming that the malware was running and investigating any possible information related to the findings from the Basic Static Analysis. LuckyCat.dll was found to be running in Process Explorer, and network traffic was captured, which confirmed connections to the ports found in the Basic Static Analysis. The Advanced Static Analysis stage involved analyzing the malware using IDA, where the code was checking whether rundll32.exe was being used to run the malware's DLL. A function called RegOpenKeyW was also found, which was used to create and open registry keys.

Lastly, the Advanced Dynamic Analysis stage involved using a combination of different tools to analyze the malware, revealing that LuckyCat was attempting to connect to a remote server at 58.158.177.102 using ports 110 and 443, and sending encrypted TCP traffic. The findings from this Reverse Engineering Project provide valuable insights into LuckyCat's behavior, which can be used to develop more effective strategies to detect and mitigate future LuckyCat attacks. The stages of analysis used in this project highlight the importance of employing various techniques to gain a comprehensive understanding of the behavior of malware.