

# 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>

50  
/ 69

Community Score

50 security vendors and no sandboxes flagged this file as malicious

e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4

Credential.dll

pedll

116.00 KB

Size

2023-03-04 10:27:03 UTC

1 month ago

DLL

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 3

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to [automate checks](#).

Basic properties

MD5

9f9723c5ff4ec1b7f08eb2005632b8b1

SHA-1

e47a821ef85d722f01f10adff227f45552e4ec73

SHA-256

e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4

Vhash

115066655d1515156058z6b7z19z31z35zb6z1

Authentihash

8cfa3d3574aa676b131d68b347361bbcec095b9226a5f742db769d2d36523a21

Imphash

068f62d1a1585dae9a2a72ce94d7d900

Rich PE header hash

60dca944da7efe09c27cfb09a3307ff7

SSDEEP

3072:Z3EKsekGvHowEu/WfW0JtLVbZDe6n7KSNMf:REKsekco0/2WEgCu/

TLSH

T1F7C37C027680C471D4BE1D38083586624BBEBD31ED746A9BA798067A9E742C0EF35F77

File type

Win32 DLL

Magic

PE32 executable for MS Windows (DLL) (GUI) Intel 80386 32-bit

TrID

Win64 Executable (generic) (32.2%) | Win32 Dynamic Link Library (generic) (20.1%) | Win16 NE executable (generic) (15.4%) | Win32 Executable (generic) (13.7%) | OS/2 Executable (generic) (6.2%)

DetectItEasy

PE32 | Compiler: EP:Microsoft Visual C/C++ (2013) [DLL32] | Linker: Microsoft Linker (14.0, Visual Studio 2015 14.0\*) [DLL32]

File size

116.00 KB (118784 bytes)

History

## 2- PEiD

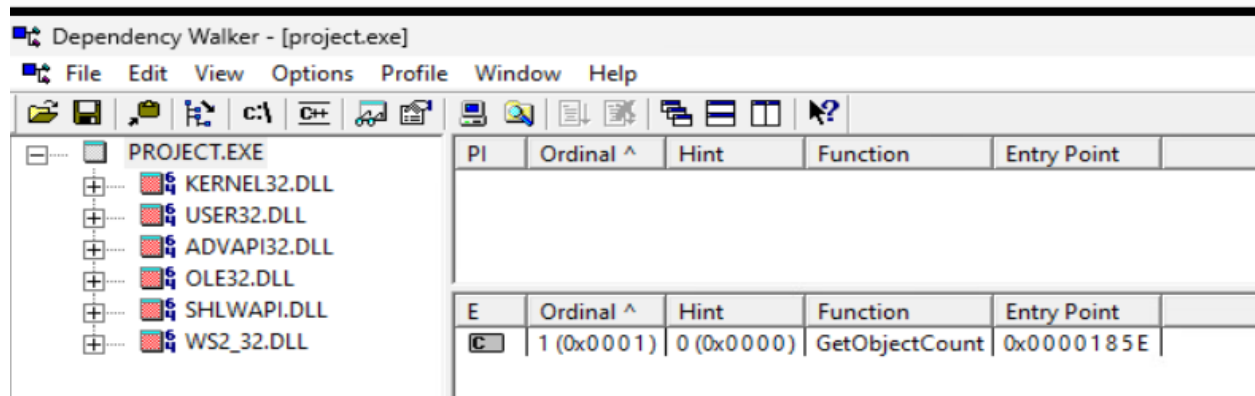
The malware, identified as Borland Delphi 3, appears to be utilizing a technique called "process replacement" to conceal itself on the system by replacing a process's memory with its own code, as evidenced by its larger virtual size compared to its row size.

Name	V. Offset	V. Size	R. Offset	R. Size	Flags
.text	00001000	000136E4	00000400	00013800	60000020
.rdata	00015000	000070AC	00013C00	00007200	40000040
.data	0001D000	000012FC	0001AE00	00000A00	C0000040
.gids	0001F000	000000D4	0001B800	00000200	40000040
.rsrc	00020000	00000358	0001BA00	00000400	40000040
.reloc	00021000	00001188	0001BE00	00001200	42000040

Borland Delphi 3.0 (???) \*

### 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, WS2\_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.

PROJECT.EXE	PI	Ordinal ^	Hint	Function
KERNEL32.DLL	N/A		127 (0x007F)	CloseHandle
USER32.DLL	N/A		182 (0x00B6)	CreateEventW
ADVAPI32.DLL	N/A		194 (0x00C2)	CreateFileW
OLE32.DLL	N/A		212 (0x00D4)	CreatePipe
SHLWAPI.DLL	N/A		219 (0x00DB)	CreateProcessW
WS2_32.DLL	N/A		232 (0x00E8)	CreateThread
	N/A		254 (0x00FE)	DecodePointer
	N/A		261 (0x0105)	DeleteCriticalSection
	N/A		266 (0x010A)	DeleteFileW
	N/A		287 (0x011F)	DuplicateHandle

PROJECT.EXE	PI	Ordinal ^	Hint	Function
KERNEL32.DLL	N/A		435 (0x01B3)	GetCPLInfo
USER32.DLL	N/A		456 (0x01C8)	GetCommandLineA
ADVAPI32.DLL	N/A		457 (0x01C9)	GetCommandLineW
OLE32.DLL	N/A		464 (0x01D0)	GetComputerNameExW
SHLWAPI.DLL	N/A		465 (0x01D1)	GetComputerNameW
WS2_32.DLL	N/A		476 (0x01DC)	GetConsoleCP
	N/A		494 (0x01EE)	GetConsoleMode
	N/A		521 (0x0209)	GetCurrentProcess
	N/A		522 (0x020A)	GetCurrentProcessId
	N/A		525 (0x020D)	GetCurrentThread
	N/A		526 (0x020E)	GetCurrentThreadId
	N/A		538 (0x021A)	GetDiskFreeSpaceExW

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.

	N/A	1347 (0x0543)	SetUnhandledExceptionFilter
	N/A	1362 (0x0552)	Sleep
	N/A	1377 (0x0561)	TerminateProcess

#### 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.

```
dalailamatrustindia.ddns.net:110  
dalailamatrustindia.ddns.net:443  
115.126.6.16:110
```

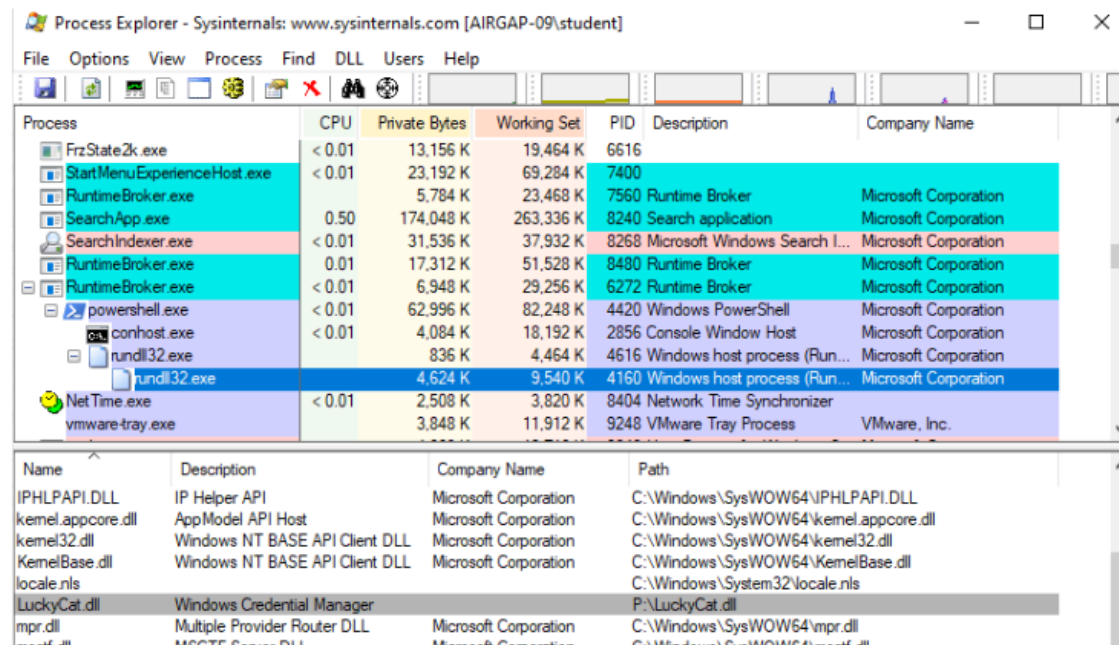
## Basic Dynamic Analysis:

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.

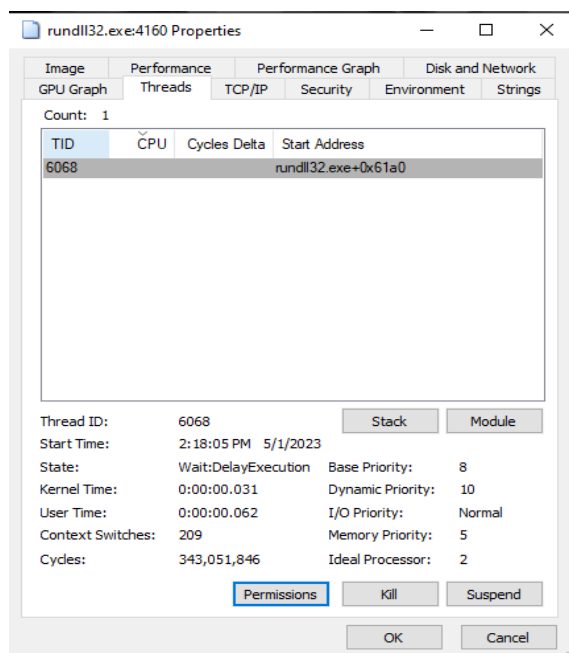
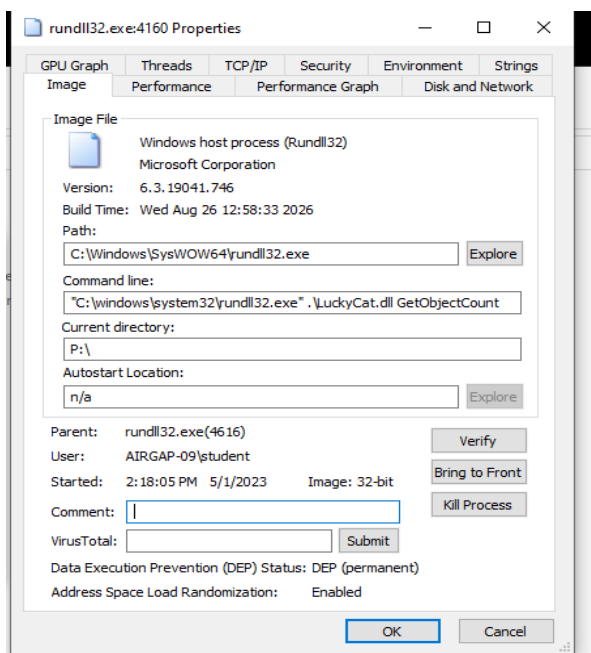


The screenshot shows Process Explorer with the 'Process' list and the 'DLL' list. The 'Process' list shows several instances of 'rundll32.exe' running. The 'DLL' list shows the loaded DLLs for the selected 'rundll32.exe' process, including 'LuckyCat.dll'.

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
FrzStateZk.exe	< 0.01	13,156 K	19,464 K	6616		
StartMenuExperienceHost.exe	< 0.01	23,192 K	69,284 K	7400		
RuntimeBroker.exe		5,784 K	23,468 K	7560	Runtime Broker	Microsoft Corporation
SearchApp.exe	0.50	174,048 K	263,336 K	8240	Search application	Microsoft Corporation
SearchIndexer.exe	< 0.01	31,536 K	37,932 K	8268	Microsoft Windows Search I...	Microsoft Corporation
RuntimeBroker.exe	0.01	17,312 K	51,528 K	8480	Runtime Broker	Microsoft Corporation
RuntimeBroker.exe	< 0.01	6,948 K	29,256 K	6272	Runtime Broker	Microsoft Corporation
powershell.exe	< 0.01	62,996 K	82,248 K	4420	Windows PowerShell	Microsoft Corporation
conhost.exe	< 0.01	4,084 K	18,192 K	2856	Console Window Host	Microsoft Corporation
rundll32.exe		836 K	4,464 K	4616	Windows host process (Run...	Microsoft Corporation
rundll32.exe		4,624 K	9,540 K	4160	Windows host process (Run...	Microsoft Corporation
NetTime.exe	< 0.01	2,508 K	3,820 K	8404	Network Time Synchronizer	
vmware-tray.exe		3,848 K	11,912 K	9248	VMware Tray Process	VMware, Inc.

Name	Description	Company Name	Path
IPHLPAPI.DLL	IP Helper API	Microsoft Corporation	C:\Windows\SysWOW64\IPHLPAPI.DLL
kernel.appcore.dll	AppModel API Host	Microsoft Corporation	C:\Windows\SysWOW64\kernel.appcore.dll
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\SysWOW64\kernel32.dll
KernelBase.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\SysWOW64\KernelBase.dll
locale.nls			C:\Windows\System32\locale.nls
LuckyCat.dll	Windows Credential Manager		P:\LuckyCat.dll
mpr.dll	Multiple Provider Router DLL	Microsoft Corporation	C:\Windows\SysWOW64\mpr.dll



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.

[illegible]

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.google syndication.com	FOUND
22:37:34	ade.google syndication.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.

43	25.464218	192.168.205.0	129.21.3.17	DNS	88 Standard query 0x01f5 A dalailamatrustindia.ddns.net
44	25.495543	192.168.205.0	129.21.4.18	DNS	88 Standard query 0x01f5 A dalailamatrustindia.ddns.net
45	25.536371	129.21.3.17	192.168.205.0	DNS	104 Standard query response 0x01f5 A dalailamatrustindia.ddns.net A 58.158.177.102
47	25.541215	129.21.4.18	192.168.205.0	DNS	104 Standard query response 0x01f5 A dalailamatrustindia.ddns.net A 58.158.177.102



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.

No.	Time	Source	Destination	Protocol	Length	Info
422	227.358514	192.168.205.0	58.158.177.102	TCP	66	49768 -> 110 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
423	227.522009	58.158.177.102	192.168.205.0	TCP	66	110 -> 49768 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1
424	227.522148	192.168.205.0	58.158.177.102	TCP	54	49768 -> 110 [ACK] Seq=1 Ack=1 Win=262656 Len=0
425	227.585118	192.168.205.0	58.158.177.102	POP	622	c: \035!E\032
426	227.748782	58.158.177.102	192.168.205.0	TCP	60	110 -> 49768 [ACK] Seq=1 Ack=569 Win=30336 Len=0
427	227.749030	58.158.177.102	192.168.205.0	POP/IMF	481	(text/html)
428	227.749030	58.158.177.102	192.168.205.0	TCP	60	110 -> 49768 [FIN, ACK] Seq=428 Ack=569 Win=30336 Len=0
429	227.749076	192.168.205.0	58.158.177.102	TCP	54	49768 -> 110 [ACK] Seq=569 Ack=429 Win=262144 Len=0
430	227.749483	192.168.205.0	58.158.177.102	TCP	54	49768 -> 110 [FIN, ACK] Seq=569 Ack=429 Win=262144 Len=0
432	227.913064	58.158.177.102	192.168.205.0	TCP	60	110 -> 49768 [ACK] Seq=429 Ack=570 Win=30336 Len=0

Wireshark - Follow TCP Stream (tcp.stream eq 5) - Ethernet0 2

.....!.....E.  
.....@.....

.....HTTP/1.1 400 Bad Request  
Date: Mon, 01 May 2023 20:31:09 GMT  
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips  
Content-Length: 226  
Connection: close  
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF/DTD HTML 2.0/EN">  
<html><head>  
<title>400 Bad Request</title>  
</head><body>  
<h1>Bad Request</h1>  
<p>Your browser sent a request that this server could not understand.<br />  
</p>  
</body></html>

08 00 45 00 PV.g.P  
b1 66 c0 a8 :(.@./  
c7 1f 50 10 ...n-h-d  
...^.....



# 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).

```
lea    eax, [ebp+Filename]
push   0FFh          ; nSize
push   eax           ; lpFilename
push   0             ; hModule
call   ds:GetModuleFileNameW
lea    eax, [ebp+Filename]
push   eax           ; lpstr
```

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.

push   72h ; 'r'	pop    eax
pop    eax	push   33h ; '3'
push   75h ; 'u'	mov    [ebp+var_18], ax
mov    [ebp+pszSrcch], ax	mov    [ebp+var_16], ax
pop    eax	pop    eax
push   6Eh ; 'n'	push   32h ; '2'
mov    [ebp+var_1E], ax	mov    [ebp+var_14], ax
pop    eax	pop    eax
push   64h ; 'd'	push   2Eh ; '.'
mov    [ebp+var_1C], ax	mov    [ebp+var_12], ax
pop    eax	pop    eax
push   6Ch ; 'l'	push   65h ; 'e'
mov    [ebp+var_1A], ax	pop    ecx
	mov    [ebp+var_10], ax
	push   78h ; 'x'
	pop    eax

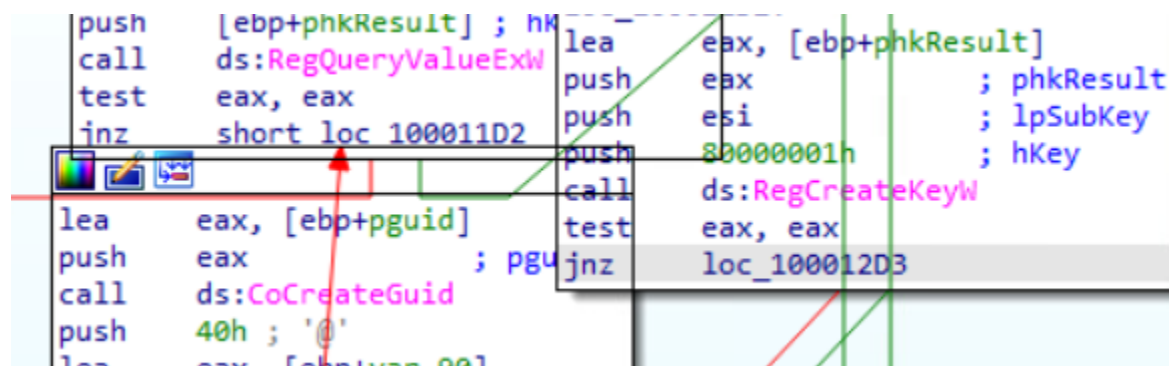
Thus, we think that the code is checking whether the (rundll32.exe) is being used to run the malware's dll. Moreover, we saw another string that has been spelled out and copied character by character and we think this method is used to avoid the detection of copying the whole string at one time. Also, 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    00000001h ; nKey
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.



```

push    [ebp+phkResult] ; hkResult
call    ds:RegQueryValueExW
test    eax, eax
jnz     short loc_100011D2
lea     eax, [ebp+pguid]
push    eax ; pguid
call    ds:CoCreateGuid
push    40h ; '0'
lea     eax, [ebp+var_10]
push    [ebp+phkResult] ; hkResult
lea     eax, [ebp+phkResult]
push    eax ; phkResult
push    esi ; lpSubKey
push    80000001h ; hKey
call    ds:RegCreateKeyW
test    eax, eax
jnz     loc_100012D3

```

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

```

```

lea     edi, [esp+7DD4h+var_E92]
mov     esi, offset a115126616110 ; "115.126.6.16:110"
push    offset aDllStartConnec ; "DLL---Start connect to %ws:%d"
lea     eax, [esp+7DDCh+OutputString]
push    eax ; dwMilliseconds
call    ds:Sleep

```

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.

The screenshot displays the x32dbg interface. The top pane shows the Symbol table with the following entries:

Address	Type	Ordinal	Symbol
71AD185E	Export	1	GetObjectCount
71AD4A39	Export	0	OptionalHeader.AddressOfEntryPoint
71AE5018	Import		kernel32.lstrcpw
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

The bottom pane shows a list of breakpoints:

Address	Disassembly	Comment
5B02580	mov edi,edi	RegCreateKeyW
5B01C30	mov edi,edi	RegSetValueExW
5B01D70	mov edi,edi	RegOpenKeyW
5B018D0	mov edi,edi	RegQueryValueExW
1AD185E	push ebp	GetObjectCount
6B967F0	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.

The screenshot shows the Windows Registry Editor with the path `Computer\HKEY_CURRENT_USER\Software\Microsoft\WAB\Resources`. A new key named `(Default)` has been created with a value type of `REG_SZ` and a data value of `(value not set)`.

Below the registry view, a command prompt window displays the following output:

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

```
08606544 | 531A31365 | return to malware 31A31365 from 033
```

```
00606E44 71A212CE Return to MatWare.71A212CE FROM ???
00606E48 00000224
```

\_\_\_\_\_

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

71E225CE	50	push eax
71E225CF	FF15 1850E371	call dword ptr ds:[<&lstrcpyw>]
71E225D5	8D46 04	lea eax,dword ptr ds:[esi+4]
71E225D8	50	push eax
71E225D9	8D85 1CD8FFFF	lea eax,dword ptr ss:[ebp-27E4]
71E225DF	50	push eax
71E225E0	E8 91030000	call malware.71E22976
71E225E5	8BC7	mov eax,edi
71E225E7	B9 38020000	mov ecx,238
71E225EC	80B405 1CD8FFFF F5	xor byte ptr ss:[ebp+eax-27E4],F5
71E225F4	40	inc eax
71E225F5	3BC1	cmp eax,ecx
71E225F8	72 5D	jb malware.71E22556

mov ecx,esi
xor byte ptr ss:[ebp+eax-27E4],F5
inc eax

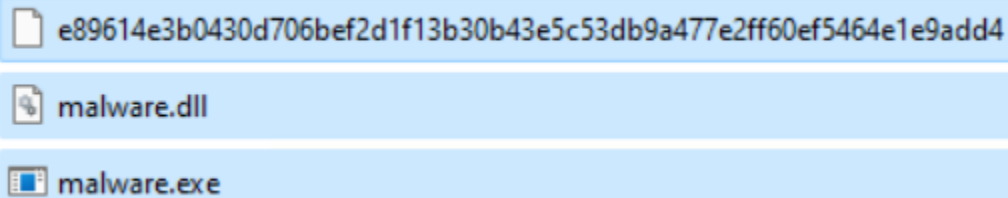
64)053896F2BFEB488898F8C647F5B0AA65  
  
  
WinDev2301Eval

Device name	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.

```
C:\Windows\System32>taskkill /F /PID 6300  
SUCCESS: The process with PID 6300 has been terminated.  
  
C:\Windows\System32>
```



e89614e3b0430d706bef2d1f13b30b43e5c53db9a477e2ff60ef5464e1e9add4  
malware.dll  
malware.exe

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

cmd.exe		6,396 K	5,216 K	7848 V
conhost.exe		1,424 K	7,560 K	7856 C
rundll32.exe		1,048 K	4,256 K	7364 V
rundll32.exe		4,388 K	8,716 K	4208 V



## **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.