The Lazarus' gaze to the world: What is behind the first stone ?

blog.telsy.com/the-lazarus-gaze-to-the-world-what-is-behind-the-first-stone

webmaster@telsy.com

// Introduction

Lazarus (aka APT38 / Hidden Cobra / Stardust Chollima) is one of the more prolific threat actors in the APT panorama. Since 2009, the group leveraged its capability in order to target and compromise a wide range of targets; Over the time, the main targets have been government and defense institutions, organizations operating in the energy and petrochemical sector in addition to those operating in financial and banking one.

The group has also a wide range of tools at its disposal; among these, it's possible to catalog [D] DoS botnets, first stage implanters, remote access tools (RATs), keyloggers and wipers. This list of malicious tools has over time supported a series of operations that have ranged from espionage to funding up sabotage.

This specific blog posts about concerns of a recent operation most likely carried out by this group and directed towards targets located in different parts of the world. However, our analysis started from a single malicious e-mail delivered against an important Italian institution operating in the banking and financial sector.

Starting from this email, we tried to trace back the path of the threat actor up to obtaining an excellent degree of visibility on what was going on.

By tracking down malware variants, the actor's operational logic and mechanisms were put in place in order to limit the spread of the second-stage payloads.

However, in this intervention, we will describe only the first phase of the kill chain; Here, the threat actor has provided for the release of two types of *payloads* based on the architecture of the victim's system as well as actions used in order to carry out a first recognition phase. Afterward, some features of the remote script that was used for managing and controlling victims will be explored. Further information about this campaign are available for our threat intelligence portal customers by referring to the investigation ***ATR:78456***.

// Vector

The threat actor, in this case, relied on a spoofed e-mail message (coming from ***e_banking***@*victim_name_domain_name*) in order to deliver to the victims a message with a malicious *Microsoft Office Word* document attached. One of these retrieved documents refers to an alleged vacant job position for the **Hindustan Aeronautics** company.

Hindustan Aeronautics Limited

Manager in Bengaluru, Karnataka

Job Role: Manager

Education Requirement: Manager

Job Locations: Bengaluru, Karnataka

Age Limit: 48-50 Years

Experience: 3 - 5 years

Salary: 60000 - 180000(per month)

Qualification in Details:

1. **Qualification Requirement:**

   Candidates are required to refer the Job Description for the details of Professional Qualification required for the respective posts.

The malicious document has two separate *first-stage* doubly ***base64*** encoded ***payloads*** included within (one for 32 and one for 64-bit systems) in addition to another doubly encoded ***base64*** *word* document that has been shown to the user.

An example of one of the ***payloads*** is shown as follows:

```
00 02 08 00 28 00 00 00 90 C3 13 80 56 46 5A 78  ....(....Ã.■VFZx
55 55 46 42 54 55 46 42 51 55 46 46 51 55 46 42  UUFBTUFBQUFFQUFB
51 53 38 76 4F 45 46 42 54 47 64 42 51 55 46 42  QS8vOEFBTGdBQUFB
51 55 46 42 51 55 46 52 51 55 46 42 51 55 46 42  QUFBQUFRQUFBQUFB
51 55 46 42 51 55 46 42 51 55 46 42 51 55 46 42  QUFBQUFBQUFBQUFB
51 55 46 42 51 55 46 42 51 55 46 42 51 55 46 42  QUFBQUFBQUFBQUFB
51 55 46 42 51 55 46 42 51 55 46 42 51 55 46 42  QUFBQUFBQUFBQUFB
51 55 46 42 51 55 46 46 51 55 56 42 51 55 45 30  QUFBQUFFQUVBQUE0
5A 6E 56 6E 4E 45 46 30 51 57 35 4F 53 57 4A 6E  ZnVnNEF0QW5OSWJn
51 6C 52 4E 4D 47 68 57 52 32 68 77 59 33 6C 43  QlRNMGhWR2hwY3lC
64 32 4E 74 4F 57 35 6A 62 55 5A 30 53 55 64 4F  d2NtOW5jbUZ0SUdO
61 47 4A 74 4E 58 5A 6B 51 30 4A 70 57 6C 4E 43  aGJtNXZkQ0JpWlNC
65 57 52 58 4E 47 64 68 56 7A 52 6E 55 6B 55 35  eWRXNGdhVzRnUkU5
56 45 6C 48 4D 58 5A 61 52 31 56 31 52 46 45 77  VElHMXZaR1V1RFEw
53 30 70 42 51 55 46 42 51 55 46 42 51 55 46 44  S0pBQUFBQUFBQUFD
5A 57 35 43 52 55 59 79 64 6A 45 76 56 6E 52 79  ZW5CRUYydjEvVnRy
4F 57 59 78 59 6D 45 76 57 44 6C 58 4D 44 52 59  OWYxYmEvWDlXMDRY
4F 46 5A 30 52 44 6C 6D 4D 57 4A 55 61 47 5A 30  OFZ0RDlmMWJUaGZ0
56 7A 49 76 4D 53 39 57 64 45 39 47 4B 30 5A 69  VzIvMS9WdE9GK0Zi
57 53 39 59 4F 56 63 77 4E 46 68 7A 56 6E 59 7A  WS9YOVcwNFhzVnYz
4F 57 59 78 59 6D 45 76 57 44 56 58 65 6E 59 31  OWYxYmEvWDVXenY1
4C 31 5A 7A 52 6D 63 78 56 6C 70 79 4C 31 67 35  L1ZzRmcxVlpyL1g5
56 33 64 58 52 47 68 57 63 33 49 35 5A 6A 46 69  V3dXRGhWc3I5ZjFi
51 6C 6C 4F 55 6C 64 78 5A 6E 67 76 56 6E 4E 47  QllOUldxZngvVnNG
5A 7A 42 47 59 6C 4D 76 57 44 6C 58 64 31 64 45  ZzBGYlMvWDlXd1dE
61 31 5A 30 64 6A 6C 6D 4D 57 4A 43 57 55 39 57  a1Z0djlmMWJCWU9W
56 7A 49 76 4D 53 39 57 63 30 5A 6E 4E 47 78 69  VzIvMS9Wc0ZnNGxi
```

Once the macro is executed, the first infection process is started using the ***AutoOpen Sub***. Variables ***dllPath*** and ***docPath*** are filled calling respectively the functions ***GetDllName()*** and ***GetDocName()*** in order to retrieve the paths from where they will be loaded later. For the first stage, it is as follows:

%USERPROFILE%"\AppData\Local\Microsoft\ThumbNail\thumnail.db

A subsequent ***LoadLibraryA*** loads dropped ***dll***. A variable named ***"a"*** is then filled with the results of the so-called ***ShowState*** function within the content of an active opened document. These instructions are the result of executing the dropped library.

// First run and persistence

The ***ShowState*** function has mainly the task of recovering the current execution path, starting the ***SetupWorkStation*** function in the same module context and ensuring persistence in the affected system.

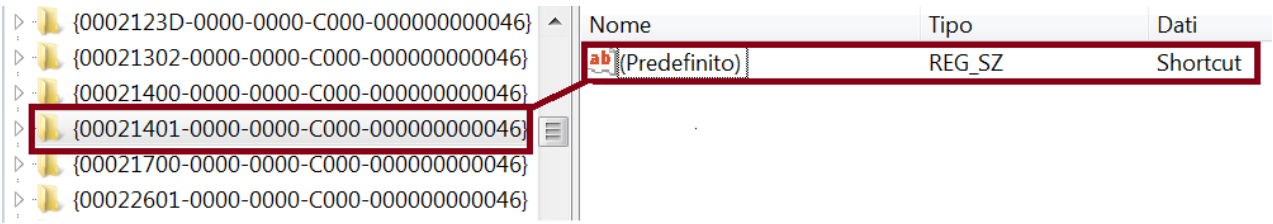
It is interesting to note how the functions ***CoInitialize*** and ***CoCreateIstance*** are used respectively to initialize the COM library and to instantiate the COM object.

```
push    ebp
mov     ebp, esp
sub     esp, 8
push    esi
push    0
call    ds:CoInitialize
lea     eax, [ebp+ppv]
push    eax
push    offset riid
push    1
push    0
push    offset rclsid
call    ds:CoCreateInstance
mov     esi, eax
test    esi, esi
```

However, in order to understand which object is being instantiated, the first argument to the ***CoCreateInstance()*** function must be inspected to extract the unique identifier (CLSID) of the COM object. A look at variable as it would look in memory is shown as follows:

```
rclsid          dd 21401h               ; Data1
                                        ; DATA XREF: sub_10007A10+1C↑o
                dw 0                    ; Data2
                dw 0                    ; Data3
                db 0C0h, 6 dup(0), 46h  ; Data4
```

Opening the ***HKEY_CLASSES_ROOTCLSID*** key gives the corresponding readable format:



On function return, a new shortcut (***lnk***) is created under the local path resulting from ***GetTempPath*** function minus "***\Local\Temp***" and plus "***\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\thumbnail.lnk***"

```
                        ; CODE XREF: sub_10007AD0+E4↑j
test    ecx, ecx
jz      short loc_10007BE6
mov     edx, 104h
sub     edx, ecx
mov     eax, 104h
sub     eax, edx
lea     ecx, [ebp+edx*2+Buffer]
push    offset aRoamingMicroso
mov     edx, 7FFFFFFFh
call    sub_10006420
```

The content of ***thumbnail.lnk*** is:

"C:\Windows\System32\rundll32.exe" "full path of module", SetupWorkStation S-6-38-4412-76700627-315277-3247 0 0 9109 1

// Implant Initialization

SetupWorkStation function of the implant is aimed at a system reconnaissance and at performing beacon of the command and control center. If the malware does not find the exact number of expected arguments in its command line, it simply quits the execution without going any further.

Inside this frame of code, a new thread is created with the starting address ***100075A0***. ***sub_10007340*** is designed to initialize external communication. It internally calls ***sub_100071F0*** that is aimed to executing operations designed for system reconnaissance.

An example of these instructions from dynamically generated *pseudo-code* is shown below:

Retrieving ***Username*** and ***CumputerName***

```
GetComputerNameW(esp7 + 0x105, (uint32_t)esp6 + 12, esi2, ebx3);
esp8 = (void*)(esp7 - 1 - 1 + 1);
esp9 = (void**)((uint32_t)esp8 - 4);
GetUserNameW(esp9 + 0x85, (uint32_t)esp8 + 12, esp7 + 0x105, (uint32_t)esp6 + 12, esi2, 0x100);
esp10 = (void*)(esp9 - 1 - 1 + 1);
ecx11 = (void*)((uint32_t)esp10 + 16);
fun_684a6770(ecx11, esp9 + 0x85, (uint32_t)esp8 + 12, esp7 + 0x105, (uint32_t)esp6 + 12, esi2, 0x100);
eax12 = (int32_t)LocalAlloc(ecx11);
esp13 = (void*)((uint32_t)esp10 - 4 + 4 - 4 - 4 - 4 + 4);
```

Retrieving ***LogicalDrives***, ***DriveTypes***

```
eax16 = (uint32_t)GetLogicalDrives(v7, v5, v3);
ecx17 = 2;
v18 = eax16;
v19 = 2;
while (1) {
    edx20 = v18 >> *(int8_t*)&ecx17 & 1;
    if (*(int8_t*)&edx20 != 1)
        goto addr_0x684a6a39_3;
    v21 = (void*)((int32_t)ebp2 - 16);
    eax22 = (int32_t)GetDriveTypeW(v21, v7, v5, v3);
    switch (eax22 - 2) {
    default:
        eax23 = (void*)((int32_t)ebp2 - 0x90);
        edx24 = 64;
        esi25 = (int32_t)"O" - (int32_t)eax23;
        do {
            if (!(edx24 + 0x7ffffbe))
                goto addr_0x684a6942_8;
```

Retrieving ***FreeSpace*** for drives

```
if (edx24) {
    addr_0x684a6949_33:
    GetDiskFreeSpaceExW((int32_t)ebp2 - 16, (int32_t)ebp2 - 0x2b8, (int32_t)ebp2 - 0x2a8,
    asm("shrd ecx, edx, 0x1e");
    asm("shrd eax, ecx, 0x1e");
    fun_684a63c0(0x100, "%", (int32_t)ebp2 - 16, (int32_t)ebp2 - 0x90, 0, 0, (int32_t)ebp2
    eax35 = 0x100;
```

Performing ***Processes Enumeration***

```
eax13 = (void*)CreateToolhelp32Snapshot();
v14 = eax13;
if (eax13 != -1 && (v15 = (void*)((int32_t)ebp1 - 0x868), v16 = eax13, eax17 = (int32_t)Process32FirstW(v16, v15), !!eax17)) {
    while (1) {
        fun_6859dfe0((int32_t)ebp1 - 0x210, 0, 0x208, v16, v15, 15, 0, v4, v2, v18, v14, 0x22c);
        v19 = v20;
        eax21 = (int32_t)CreateToolhelp32Snapshot(8, v19, v16, v15);
        edi22 = eax21;
        if (edi22 == -1) {
            addr_0x684a6591_3:
            eax23 = 0x40000;
            ecx24 = ebx25;
        } else {
            fun_6859dfe0((int32_t)ebp1 - 0x634, 0, 0x424, 8, v19, v16, v15, 15, 0, v4, v2, v26);
            v27 = (void*)((int32_t)ebp1 - 0x638);
            v28 = edi22;
            eax29 = (int32_t)Module32FirstW(v28, v27, 8, v19, v16, v15);
```

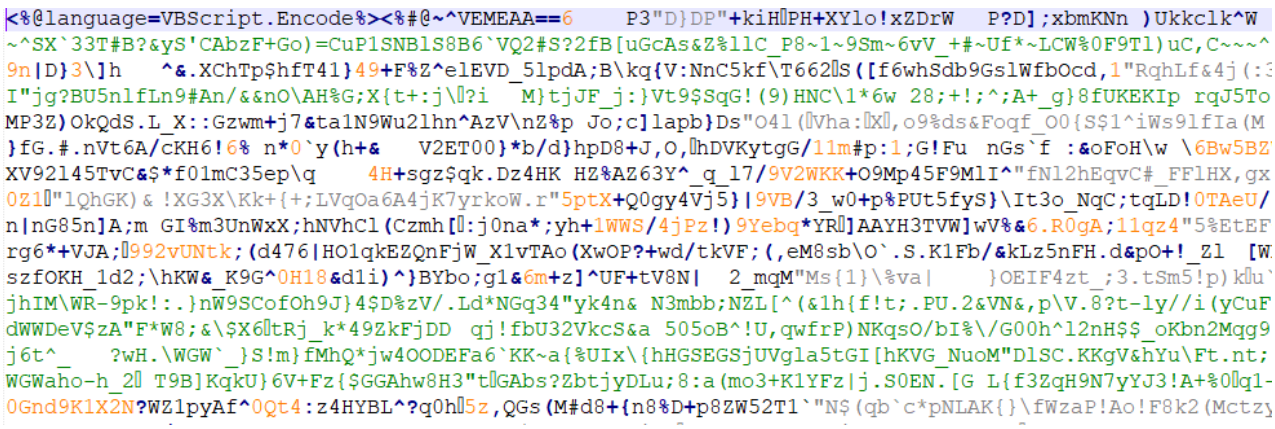
The collected information is then compressed and encrypted. Subsequent HTTP request is prepared in order to send data to command and control. Communications make use of HTTP protocol and POST method. “**ned**“, “**gl**” and “**hl**” parameters will be used in order to interact with remote command and control script that are used to handle victims and to deliver the second stage *payload*. A code frame regarding the functions used for HTTP communication is reported as follows:

```
call    ds:WinHttpOpenRequest
mov     esi, eax
cmp     esi, ebx
jz      loc_10006EA7
cmp     [ebp+arg_C], ebx
jz      short loc_10006C91
push    4
lea     ecx, [ebp+Buffer]
push    ecx
push    1Fh
push    esi
mov     [ebp+Buffer], 3300h
call    ds:WinHttpSetOption
```

// Behind the first stone

We had the opportunity to analyze what the actor did in the backend in order to manage the victims of the first stage implanter that has been described. The remote script, at least as far as observed, is copied into legitimate compromised sites. It also includes the possibility to decide if and when the second level *payload* is to be released and works through blacklists and whitelists in order to protect the final backdoor from unwanted spread.

It looks like a heavily obfuscated VBScript artifact. Here an extract from the original retrieved code:



After retrieving the original instructions set, it has been possible to deeply understand the working logic behind; The remote script works mainly through ***Request.Form*** variables that are filled when receiving beacons from victims and by local variables named as following:

1. ***strworkdir****:* The working folder within the compromised ***wwwroot***.

2. ***strlogpath***: The path to the file used in order to log victims' data. In this case a fake .mp3 file

3. ***strwhitefile****:* The path to the file used in order to store whitelisted victims IP address. In this case, a fake .mp3 file.

4. ***strblackfile****:* The path to the file used in order to store the blacklisted IP address. In this case, a fake .mp3 file.

Parameters "**gl**" and "**hl**" are used respectively to retrieve system info about victims and OS architecture. On the basis of what we have collected, the log file mapped by ***strlogpath*** variable is then updated with a new row comprising *victim IP address*, *victim system info*, *request timestamp* and *adopted case* in handling the victim.

The ***cases*** that have been designed by the threat actor can be four on the basis of interest for the victim:

1. ***case_1_64/86***: MD5 of IP address that made the request is on whitelist. The actor has selected the victim to be infected with a second-stage payload. ***TorisMa_x64/86*** *payload* is then released to the victim.
2. ***case_2_64/86***: MD5 of IP address that made the request is on blacklist. The actor wants to prevent the spreading of the second stage payload to that IP address. ***Doris_x64/86*** (non-sense chars) *payload* is then released to the victim.
3. ***case_3****:* The victim results of particular interest for the threat actor on the basis of retrieved system info (identified with a value of **24** of "**ned**"). Second stage payload is not yet delivered.
4. ***case_4:*** The victim results of no particular interest for the threat actor. no previous condition has been met. Second stage payload is not yet delivered.

Below, the primary construct used to manage what is received by the backend script:

```
If ipOK(stRWhiTeFile,strMD5IpAddr)=1 AnD InsTr(sTrOsBit,"1")>0 Then r=WriTeLine(strLogPath,"case_1_64")
sTrResData=strBaSE64_ToriSma_x64
ElsE If IpoK(strWHiteFiLe,strMD5IpAddr)=1 And InStr(strOsBit,"0")>0 ThEN r=WRITeLine(strLogPath,"case_1_86")
strResDATa=strBase64_tORIsMa_x86
ELse If IpOk(sTrBlacKFile,strMD5ipAddr)=1 ANd InSTr(sTrOsBiT,"1")>0 Then r=writeLine(strLogPath,"case_2_64")
strResDAta=strBAse64_Doris_x64
Else If IpOK(sTrBlaCkFile,strMD5ipAddr)=1 AnD Instr(stROSBit,"0")>0 then r=WriTeLine(strLogpatH,"case_2_86")
strResData=stRBase64_doRis_x86
ElsE if inStR(strCOnditiOn,"24")>0 Then r=WrIteLine(stRlogPAth,"case_3")
Response.StaTuS="404 File Not Found"
ELse r=WriTeLIne(stRlogPath,"case_4")
resPOnse.Status="404 File Not Found"
End If
```

// Victimology

According to the visibility obtained so far, we asses with a high degree of confidence that this campaign is mainly directed against research/defense sector and financial / payments institutions. Other types of sectors are obviously not to be excluded on the basis of actor interests. Most of the malicious activities associated with the examined malware set are limited to the Indian region. However, organizations of other countries as well are inside of Lazarus' interests. Here there is an exhaustive geographical map where it is possible to observe actions attributable to this specific threat (note that these malicious actions may not have led to a current active infection but could be only limited to infection attempts):

// Conclusions

In this case, the Lazarus group targets research / defense and financial organizations mainly in the same region where the security community has recently attributed an attack from the same group against a nuclear power plant. However, it has also been noted that the actor has extended its interests to other regions of the world, including Italy. Furthermore, we have observed an info-

gathering implanter used to quickly identify interesting targets and we have exposed the use of a backend script designed to handle the victims and limit the spread of second-stage *payloads* only to wanted ones.

// MITRE ATT&CK Techniques

[+] T1193 – Actor relies on spear-phishing as infection vector

[+] T1002 – Actor compresses and encrypts data

[+] T1132 – Actor encodes data

[+] T1023 – Actor relies on shortcuts to achieve persistence

[+] T1060 – Malware maintain persistence through Start menu folder

[+] T1071 – Actor relies on standard application layer protocol for C2 coms

[+] T1043 – Actor uses common ports to communicate

// Indicators of Compromise

SHA256: *b018639e9a5f3b2b9c257b83ee51a3f77bbec1a984db13d1c00e0CC77704abb4*

SHA256: *adf86d77eb4064c52a3e4fb3f1c3218ee2b7de2b1780b81c612886d72aa9c923*

SHA256: *1a172d92638e6fdb2858dcca7a78d4b03c424b7f14be75c2fd479f59049bc5f9*

SHA256: *ec254c40abff00b104a949f07b7b64235fc395ecb9311eb4020c1c4da0e6b5c4*

SHA256: *26a2fa7b45a455c311fd57875d8231c853ea4399be7b9344f2136030b2edc4aa*

Domain name (compromised): *curiofirenze[.]com*

IP Address: 193.70.64.163

File: *%USERPROFILE%"\AppData\Local\Microsoft\ThumbNail\thumnail.db*

File: *%APPDATA% \Microsoft\Windows\Start Menu\Programs\Startup\thumbnail.lnk*

// Artifacts detection rules

YARA detection rule for unpacked dll implant is available here

Third-party freely available rules for detecting executables that have been encoded with *base64* twice are here