Panda’s New Arsenal: Part 2 Albaniiutas

insight-jp.nttsecurity.com/post/102gkfp/pandas-new-arsenal-part-2-albaniiutas

Hiroki Hada

はじめに

前回のブログでは、TA428が使用する新たなマルウェアであるTmangerについて紹介しました。Tmangerは横展開後に実行され、典型的なRATとして活動するマルウェアです。Tmangerにはたくさんのバージョンがありますが、それとは様々な実装が異なるものの、明らかに関連していると思われるマルウェアが存在します。今回はそうしたTmangerに関連するマルウェアの中から、Albaniiutasを紹介します。

Albaniiutas

2020年7月、私達はこれまでのTmangerと若干挙動が異なるマルウェアを発見しました。これまでに発見したTmangerは全て極めて類似した実装でしたが、今回発見したマルウェアはそれとは明らかに異なるものです。私達は発見時のファイル名から、このマルウェアをAlbaniiutasと呼んでいます。

しかし、標的やタイムスタンプ（IOCの章で後述します）、リソース領域の命名規則、3つのパートから成り立つこと、マルウェア内で使用されているディレクトリ名がTmangerと類似しています。これらのことからこのマルウェアは私達が観測したTmangerの亜種か、あるいは同一の作成者によって作られたものであると考えています。

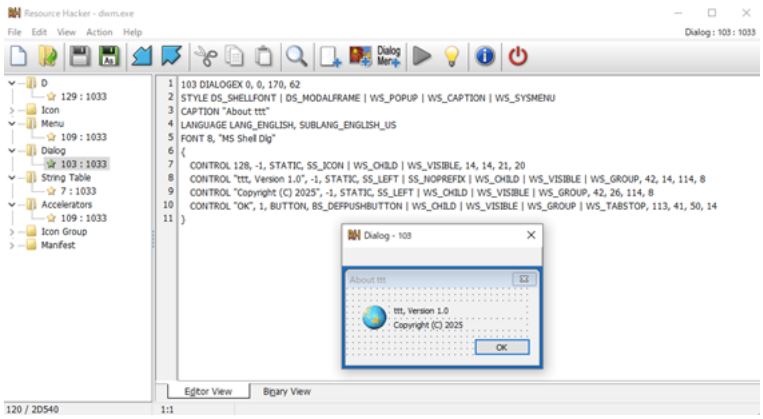


図 1 Tmangerのリソースデータ

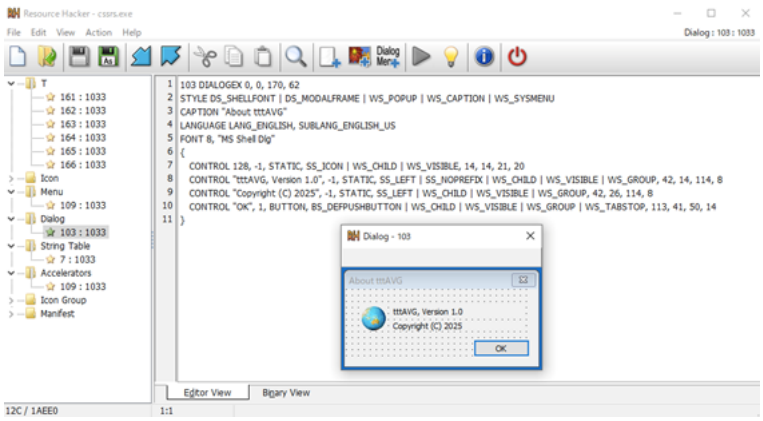


図 2 Albaniiutas のリソースデータ

Analysis Result

攻撃はalbaniiutas.rarというファイルから始まります。このRARファイルを解凍すると utas.xlsx .exeという名前のファイルが得られます。XLSXファイルのように偽装されていますがEXEファイルです。このファイルを実行すると、リソース領域に保存されているXLSXファイルとEXEファイルが作成され、開かれます。

作成されたXLSXファイルにはモンゴルのCitizens' Representative Huralのメンバーの連絡先が書かれていました。このことから、攻撃者はモンゴルの政治系の組織を標的としていると考えられます。これはTA428の特徴と一致します。

EXEファイルはcssrs.exeという名前で C:\MSBuild\WindowsUpdate\S-1-2\ というディレクトリに作成されました。このファイルを起点にいくつかのファイルが生成されますが、その役割はTmangerと極めて類似しています。

Albaniiutas File	Tmanger Type
cssrs.exe	SetUp
(2nd) vjsc.dll, XpEXPrint.dll	MloadDll
ClientX.dll	Client

各ファイルは図 3のように実行されていきます。

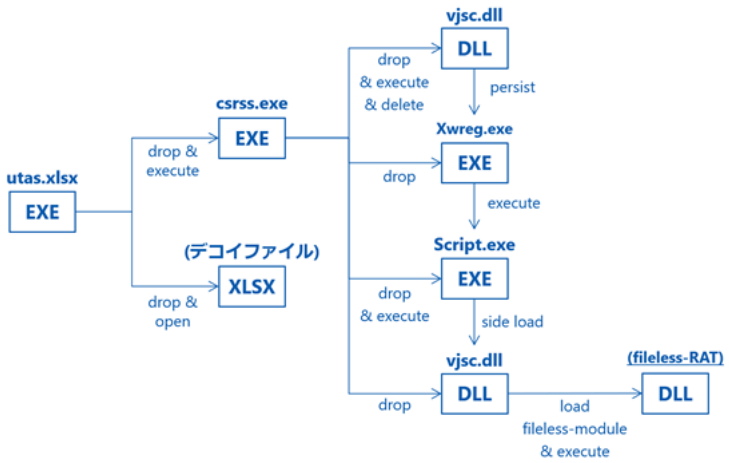


図3 全体の流れ（Adminではない場合）

cssrs.exe

まずCreateEventWで {F14E0EF3-E26A-4551-8F84-08E738AEC912} という名前のイベントを作成します。これはTmangerが作成するイベント名の規則に一致します。

その後、IsUserAdminでユーザの権限を確認し、処理を分岐します。

Adminの場合

まずRC4を使ってデータをデコードします。使用された暗号鍵は **L!Q@W#E$R%T^Y&U*A|}t~k** です。デコードされたデータは XpEXPrint.dll で、以降で作成するDLLの名前となっています。

次に、リソース領域から 162 というデータを取得します。このデータをWinAPIのCrypto系のAPIを使って、AES-256でデコードします。そのときの鍵を生成するために e4e5276c00001ff5 という文字列が使用されます。復号すると、deflateされたデータになります。それをinflateし、XpEXPrint.dllという名前でSystem32へ保存します。

その後、同様にリソース領域からデータを取得します。取得するデータは 163 と 165 です。取得したデータは **L!Q@W#E$R%T^Y&U*A|}t~k** を鍵としてRC4でデコードされます。163は vjsc.dll というファイル名で、165は Scrpt.exe という名前で System32へ保存されます。

ファイルへ書き込むとき、vjsc.dllには C:\Users\power\AppData\Local\Microsoft\Internet Explorer\CXXX.dll という文字列が含まれていますが、その部分を自分自身のファイルパスに書き換えます。このパスは攻撃者がマルウェアを作成する際に使用している可能性があります。続いて、ファイルの更新時刻を10年前の値に設定します。これはSystem32でファイルが目立たないようにするためだと考えられます。

```
0x004018fb      lea eax, [lpSystemTime]
0x004018fe      push eax            ; LPSYSTEMTIME lpSystemTime
0x004018ff      call dword [GetSystemTime] ; 0x411048 ; VOID GetSystemTime(LPSYSTEMTIME lpSystemTime)
0x00401905      mov eax, 0x7da      ; 2010
0x0040190a      mov word [lpSystemTime], ax
0x0040190e      lea eax, [lpLastAccessTime]
0x00401911      push eax            ; LPFILETIME lpFileTime
0x00401912      lea eax, [lpSystemTime]
0x00401915      push eax            ; const SYSTEMTIME *lpSystemTime
0x00401916      call dword [SystemTimeToFileTime] ; 0x411040 ; BOOL SystemTimeToFileTime(const SYSTEMTIME
0x0040191c      push 0              ; HANDLE hTemplateFile
0x0040191e      push 0x80           ; 128 ; DWORD dwFlagsAndAttributes
0x00401923      push 3              ; 3 ; DWORD dwCreationDisposition
0x00401925      push 0              ; LPSECURITY_ATTRIBUTES lpSecurityAttributes
0x00401927      push 0              ; DWORD dwShareMode
0x00401929      push 0x10000000     ; DWORD dwDesiredAccess
0x0040192e      lea eax, [lpOutputString]
0x00401934      push eax            ; LPCWSTR lpFileName
0x00401935      call dword [CreateFileW] ; 0x411094 ; HANDLE CreateFileW(LPCWSTR lpFileName, DWORD dwDesi
0x0040193b      mov edi, eax
0x0040193d      cmp edi, 0xffffffff
0x00401940      je 0x401957
0x00401942      lea eax, [lpLastAccessTime]
0x00401945      push eax            ; const FILETIME *lpLastWriteTime
0x00401946      push eax            ; const FILETIME *lpLastAccessTime
0x00401947      push eax            ; const FILETIME *lpCreationTime
0x00401948      push edi            ; HANDLE hFile
0x00401949      call dword [SetFileTime] ; 0x41109c ; BOOL SetFileTime(HANDLE hFile, const FILETIME *lpCr
0x0040194f      push edi
0x00401950      call ebx
```

図 4 ファイルの更新時刻を書き換え

そしてScrpt.exeをShellExecuteExWで実行します。Scrpt.exeはMicrosoftのVisual J#のコマンドラインツールで、電子署名が付与されている正規バイナリです。Scrpt.exeは実行時に同じディレクトリにある vjsc.dll をサイドロードし、VJSCCommandLineCompile というエクスポート関数を実行します。

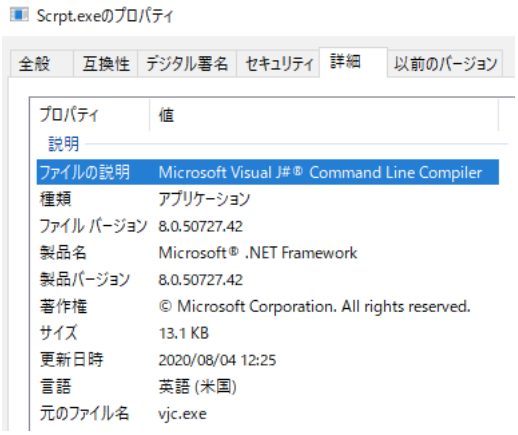


図 5 Scrpt.exeのプロパティ

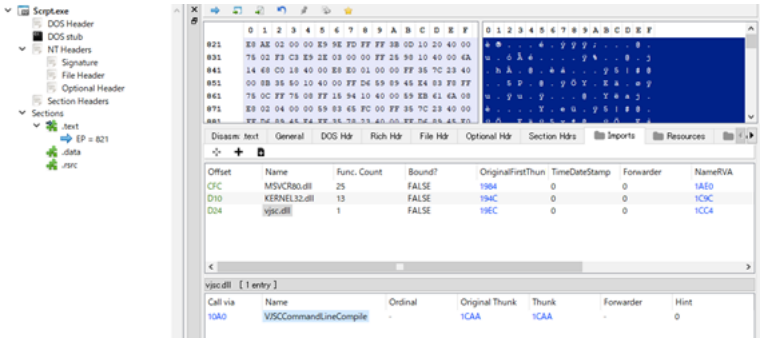


図 6 Scrpt.exeが呼び出すvjsc.dllの関数

vjsc.dllは、まずXOR 0x88でデータをデコードします。デコードされた文字列は以下のとおりです。

- DFS Replication
- FTP Publishing Service
- ReadyBoost
- Software Licensing
- SL UI Notification Service
- Terminal Services Configuration
- Windows Media Center Extender Service
- Windows Media Center Service Launcher
- SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost
- RegOpenKeyExA
- netsvcs
- RegQueryValueExA
- OpenSCManagerA
- %SystemRoot%\System32\svchost.exe -k netsvcs
- MACHINE\SYSTEM\CurrentControlSet\Services\
- RegSetValueExA
- GetSystemDirectoryA

その後、XpEXPrint.dllをランダムな4文字のファイル名でコピーします。そして、先にデコードした文字列を使って、サービスとして登録し、起動します。

Adminではない場合

リソース領域から 161 と 164 と 165 を読み込み、RC4でデコードします。そのときに使用する暗号鍵は **L!Q@W#E$R%T^Y&U*A|}t~k** です。161は vjsc.dll、164は Xwreg.exe、165は Scrpt.exe というファイル名で AppData\Local\Microsoft\Internet Explorer へ書き込まれます。

その際、vjsc.dllは C:\Users\Waston\AppData\Local\Microsoft\Internet Explorer\FindX.exe を、Xwreg.exeは C:\Users\Waston\AppData\Local\Microsoft\Internet Explorer\WSMprovhost.exe という文字列を含んでいますが、これらを自分自身のパスに置き換えます。 C:\Users\Waston はTmangerのPDBのパスと一致しており、攻撃者がマルウェアを作成する際に使用している環境であると推測できます。その後、ファイルの更新時刻を10年前の値に設定します。

その後、ShellExecuteExWでScrpt.exeを実行し、同じディレクトリに存在する vjsc.dll をサイドロードします。

vjsc.dllの内部名は RegAdd.dll で、その名のとおり同じディレクトリに存在する Xwreg.exe をレジストリのCurrentVersion\Runを使って自動起動の設定を行います。

次にcssrs.exeはvjsc.dllを削除し、リソース領域から 162 を読み込み、AES-256でデコードします。デコードする際はWinAPIのCrypto系のAPIを使用します。鍵を生成する文字列は e4e5276c00001ff5 です。得られたデータは vjsc.dll という名前で先程と同じようにAppData\Local\Microsoft\Internet Explorer へ書き込まれます。

最後に、再びShellExecuteExWでScrpt.exeを実行し、同じディレクトリに存在する vjsc.dll をサイドロードします。

XpEXPrint.dll

Adminの場合に起動される XpEXPrint.dll と、Adminではない場合にScrpt.exe経由で実行される vjsc.dllは同一のファイルです。

ServiceMainから起動した場合、 {A24D0DC3-E26A-4551-8F84-08E738AEC718} というイベントを作成します。その後の挙動は同一です。

まず、リソース領域に保存されている T というデータの 104 を読み込みます。これを e4e5276c00001ff5 をいう文字列を鍵の生成データとしてAES-256でデコードします。そうすると、DLLファイルが得られ、それをロードして実行します。そのDLLの内部名は ClientX.dll でした。

ClientX.dll

ClientX.dllはTmangerのClientと同様にRAT本体です。まず、**L!Q@W#E$R%T^Y&U*A|}t~k** を鍵としたRC4でconfigをデコードします。得られたconfigは以下のとおりです。

- http[:]//go.vegispaceshop[.]org/shop.htm
- 0
- AppData
- Roaming
- 0.0.0.0:

同様にUser-AgentをRC4でデコードします。その後、gethostnameでホスト名を取得し、User-Agentに追加します。以下のようなUser-Agentが使用されます。

Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:71.0) Gecko/20100101 Firefox/71.0 [HOSTNAME])

RC4のデコードが完了した後、デコードされたURLに対してアクセスし、ダウンロードしたHTMLファイルから新しいC&Cサーバーの IPアドレスをデコードします。デコードされたC&Cサーバーの IPアドレスは、この後説明する、感染端末の情報送信やコマンド実行をする際に使用されます。

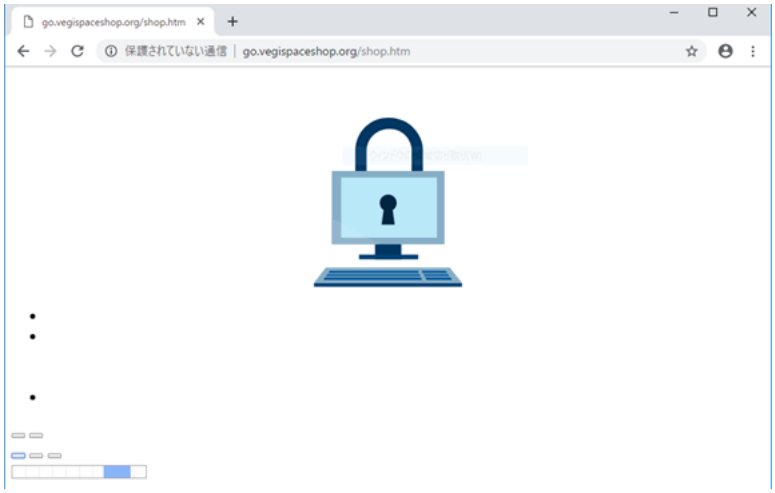
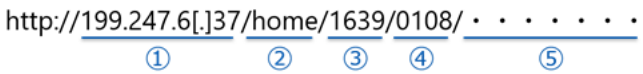


図 7 ダウンロードされたHTMLファイル

```
import sys
def main():
    input_bin = read_htmlfile()
    extracted_data = extract_data(input_bin)
    result = decode_bin(extracted_data)
    with open("result.bin","wb") as f:
        for b in result:
            f.write(b.to_bytes(1,byteorder="big"))

def extract_data(input_bin):
    index = 0
    data = list()
    while index+2 < len(input_bin):
        if input_bin[index] != 0x3E or input_bin[index+1] != 0x9:
            index = index + 1
            continue
        index = index + 2
        sub_bin_ary = get_unitl_cr(input_bin[index:])
        if len(sub_bin_ary) == 0:
            return list()
        data = data + sub_bin_ary
        index = index + len(sub_bin_ary) + 1
    return data

def get_unitl_cr(bin_ary):
    if len(bin_ary) == 0 or bin_ary[0] == 0x0d:
        return list()
    last_index = len(bin_ary) -1
    ret = list()
    for ii in range(0,len(bin_ary)-1):
        if bin_ary[ii] == 0x0d:
            break
        if bin_ary[ii] == 0x0a:
            return list()
        ret.append(bin_ary[ii])



    return ret

def decode_bin(bin_ary):
    ret = list()
    for ii in range(0,len(bin_ary)//8):
        ret.append(0)
    start_index = 0
    for ii in range(7,-1,-1):
        kk = 0
        while kk < (len(bin_ary)//8):
            val = bin_ary[kk*8+start_index]
            ret[kk] += ( val & 0x1 ) << ii
            kk += 1
        start_index = start_index + 1
    return ret

def read_htmlfile():
    with open(sys.argv[1],"rb") as f:
        return f.read()

if __name__ == "__main__":
    main()
```

図 8 新しいC&Cサーバーをデコードするロジック

IPアドレスのデコード後、ClientX.dllは感染端末の情報やコマンド処理に必要な情報をC&Cサーバーに送信します。送信されるURLは下記のような形式となっており、URLのパス部に感染端末の情報やコマンド処理の通信暗号化に必要な情報が含まれています。URLのパス部の末尾は、「hostnameコマンドの実行結果」と「CoCreateGuid()で生成されたGUID」と「GetTickCount()の戻り値」がAES256で暗号化され、Base64でエンコードした文字列です。この暗号化された文字列を復号するコードは以下の通りです。以下のコードの変数encryptedに暗号化された文字列を設定することで、復号することができます。



① 新しく取得したC&Cサーバーの IPアドレス

② “home”という固定の文字列
③ システム時間を基にしたランダムな値であり、 コマンド実行の際に利用される
④ ⑤の文字数を10進数で表した数値
⑤ 暗号化された文字列

図9 感染端末の情報やコマンド実行の際に必要な情報を送信する際のURL

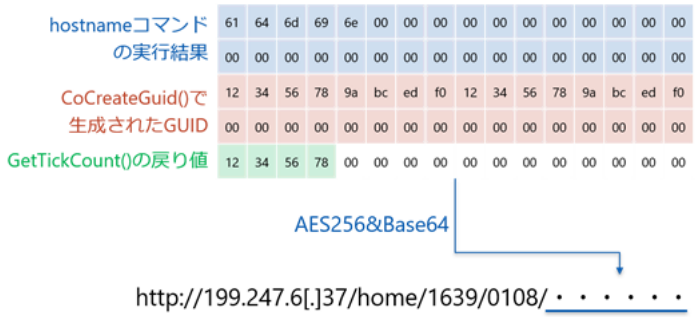


図10 URLパスに含まれる暗号化された文字列のデータ構造

```
import hashlib
from Crypto.Cipher import AES
from binascii import a2b_hex
import base64
BS = 16
def aes_decrypt(hash_seed,encrypted_data):
        aes_key = hashlib.sha256(hash_seed).hexdigest()
        encrypted_data = pcks_pad(encrypted_data)
        cipher_aes = AES.new(key=a2b_hex(aes_key),mode=AES.MODE_CBC,IV=b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
        return cipher_aes.decrypt(encrypted_data)

def pcks_pad(raw):
        result = list(raw)
        pad_ch = BS - len(result) % BS
        for ii in range(0,pad_ch):
                result.append(pad_ch)
        return bytes(result)
hash_seed = "303330343237366366346633313334350000000000000000000000000000000000"
encrypted =
"426d62484c4d2f495a734a6c444a716d335037784e6451534c6a534761636a6954694864454d4636776a6347366e71692f43457a52366e5a5834
encrypted = a2b_hex(encrypted)
plain = aes_decrypt(a2b_hex(hash_seed),base64.b64decode(encrypted))
print("DecryptedData:" + bytes.hex(plain))
```

図 11 URLのパス部に含まれる暗号化された文字列を復号するコード

その後、ClientX.dllはC&Cサーバーから受信したデータを基にコマンドを実行します。cmd.exeの実行の他、ファイルのアップロードやダウンロードを実行できます。

コマンドID	オプション	説明
(コマンド)	コマンドの引数	cmd.exeを用いてコマンドを実行し、結果をC&Cサーバーに返す。
-upload	• 感染端末上のファイルパス • アップロード時のURLのパス部	ファイルをアップロードする

-download	• ダウンロードURL • 保存先のファイルパス	ファイルをダウンロードする
-exit		何もしない

図 12 コマンドのリスト

コマンド実行する際の通信はAES256で暗号化されており、ClientX.dllは下記のような処理でデータを復号していました。CryptHashData()の第2引数には図 8で説明したGUIDが指定されます。

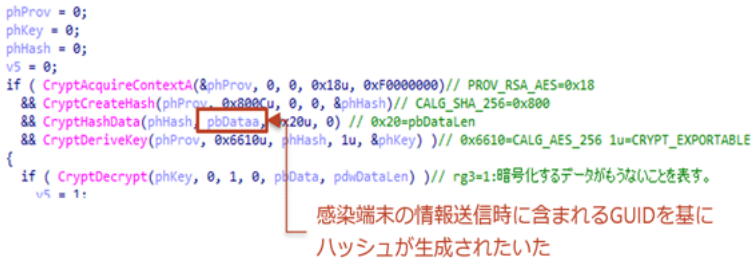
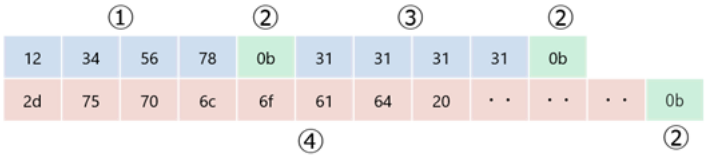


図 13 コマンド実行時の通信を復号する処理

コマンド実行時に受信するデータの形式は下記のようになっています。また、各コマンドについて、受信するデータの例を示します。



① コマンドを複数回実行する場合、前回と異なる値を指定しないとコマンドが実行されない。

② 区切り文字

③ 図 7の③の値と一致していない場合、コマンドが実行されない。

④ コマンドIDとコマンドのオプションを意味しており、スペースで区切られている。

図 14 コマンド実行時に受信するデータの形式

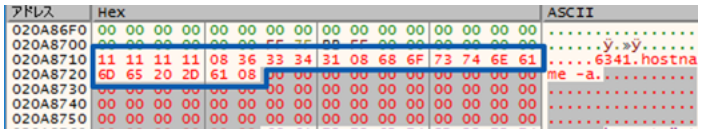


図 15 cmd.exeを実行する場合のデータ例

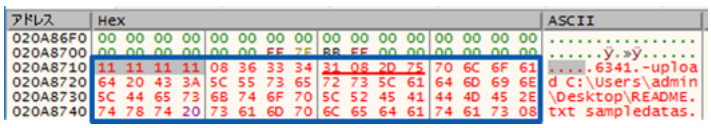


図 16 ファイルをアップロードする場合のデータ例

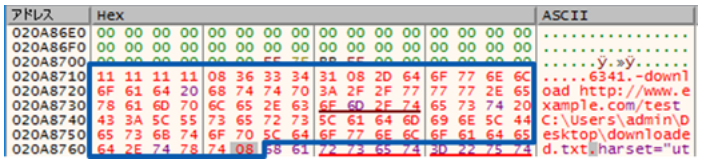


図 17 ファイルをダウンロードする場合のデータ例

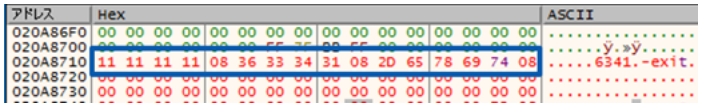


図 18 何もしない場合のデータ例

さいごに

今回はTmangerに関連すると思われるマルウェアの中から、Albaniiutasを紹介しました。AlbaniiutasはTmangerに類似していますが、C&Cサーバーとの通信処理やコマンド処理などの細部では大きな差が存在します。コンパイルタイムやリソース名を比較することで分かりますが、AlbaniiutasはTmangerと同じ人物によって実装されたマルウェアであり、Tmangerを参考にして作成された可能性が高いです。このようなマルウェアはAlbaniiutas以外にも存在し、今後様々な攻撃に利用される恐れがあります。今後の動向に注視すべきでしょう。次回はAlbaniiutasと同じようにTmangerに関連すると思われるマルウェアの中から、Smanagerについて紹介します。

IOC

C&C Server

- http[:]//go.vegispaceshop[.]org/shop.html
- http[:]//209[.]250.239.96/index.html
- http[:]//209[.]250.239.96/home/

File Hash

SHA256	Timestamp
5eb4a19fbd25ecdabf2a456a23251f13fa938400cb32cfe87a62e8c168f9b841	(UTC) 2025-12-10 23:12:16
29152de94199d77b0da9fc89d5b80bd4692f4aadf9e8362a2aee0a3b455c4e76	(UTC) 2025-12-10 23:12:21
fd43fa2e70bcc3b60236366756049422 9287bf4716638477889ae3f816efc705	(UTC) 2025-12-21 03:43:51
cf36344673a036f5a96c1c63230c9c15bb5e4f440eafd4ba0dc01d44bb1df3bf	(UTC) 2025-12-22 13:21:31
d94f404b2b5bafa0d9ce66219b2684186715f5ef20a69f036a06d465177d5769	(UTC) 2025-12-23 10:06:11
71750c58eee35107db1a8e4d583f3b1a918dbffbd42a6c870b100a98fd0342e0	(UTC) 2025-12-23 11:00:09