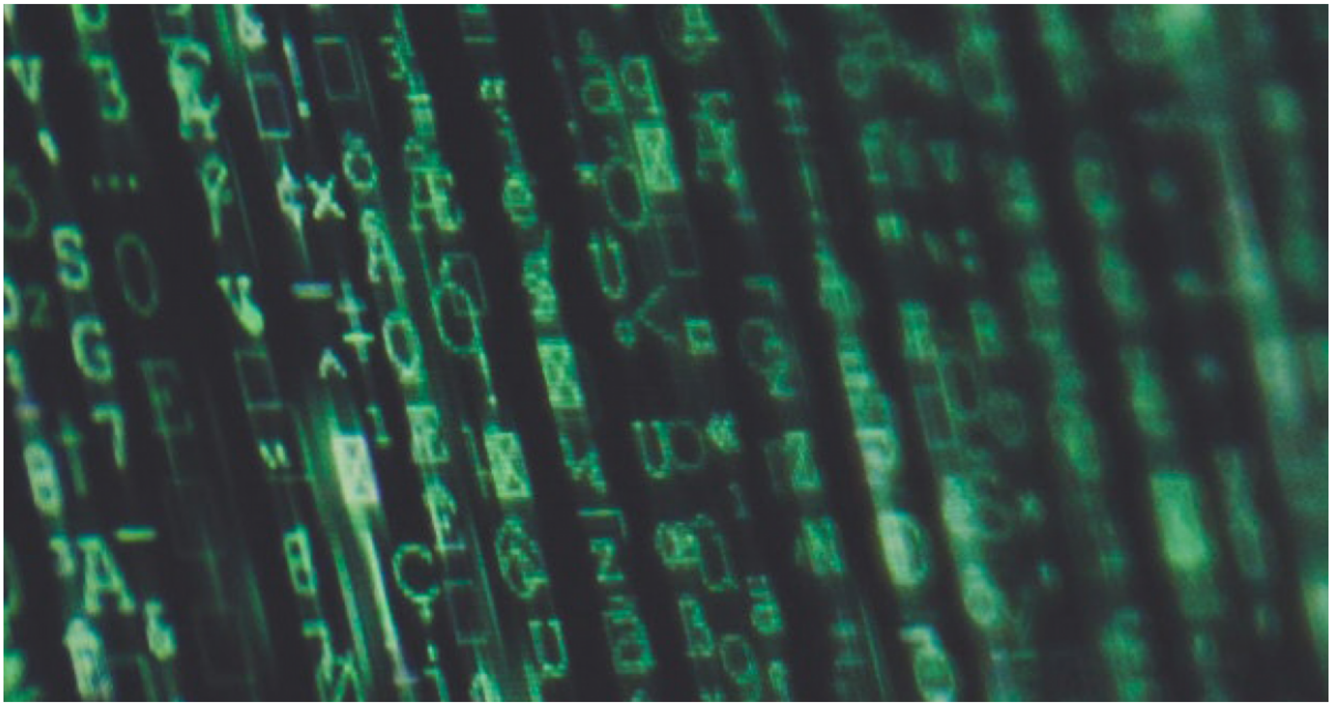
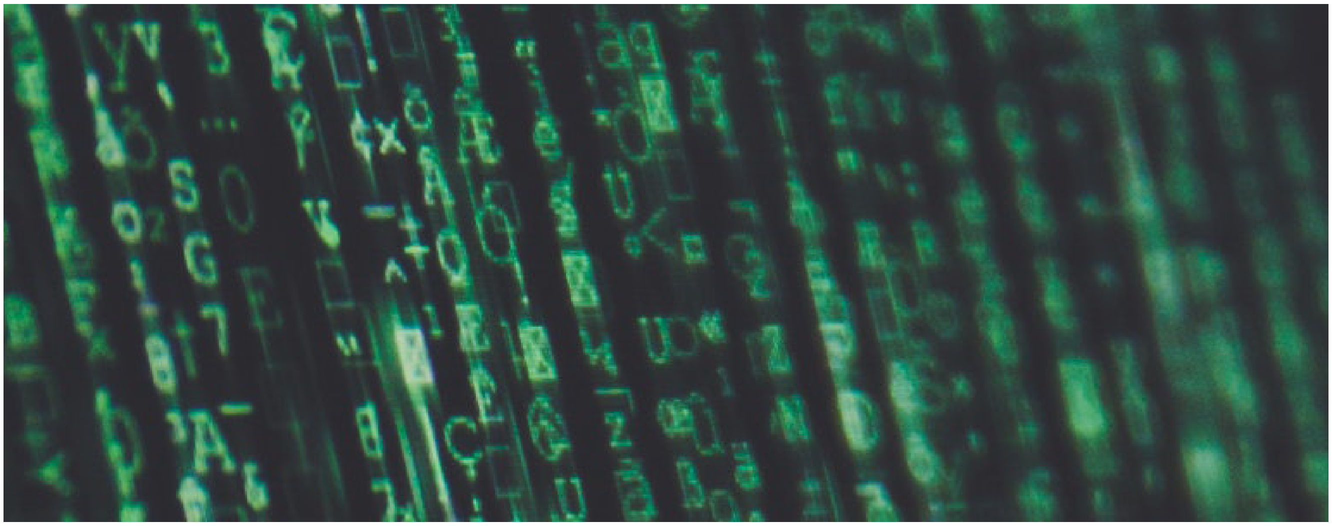
DanaBot Gains Popularity and Targets US Organizations in Large Campaigns

proofpoint.com/us/threat-insight/post/danabot-gains-popularity-and-targets-us-organizations-large-campaigns

October 2, 2018



DanaBot Gains Popularity and Targets US Organizations in Large Campaigns



October 02, 2018 Proofpoint Staff

Overview

Proofpoint researchers first discovered DanaBot in May of 2018 [1], describing its use by a single actor targeting Australian organizations. As we predicted at the time, other threat actors targeting Europe and North America have since adopted the banking Trojan, increasing its footprint and taking advantage of its extensive anti-analysis features. In this blog we describe a campaign affecting organizations in the United States and present new reverse engineering analysis of DanaBot.

Recent DanaBot Campaigns

Our colleagues at ESET recently blogged about DanaBot campaigns and described the latest expansion of targeted countries to include Poland, Italy, Germany, and Austria [2]. We have also observed several campaigns since May targeting Australia. Finally, at the end of September, an actor that typically targets the United States with daily campaigns distributing the Panda banking Trojan switched to delivering DanaBot for a day.

Hancitor Campaign

On September 26, Proofpoint researchers observed a campaign with hundreds of thousands of email messages targeting US recipients. The emails used an eFax lure (Figure 1) and contained a URL linking to the download of a document containing malicious macros (Figure 2). The macros, if enabled by the user, executed the embedded Hancitor malware [3], which, in turn, received tasks to download two versions of Pony stealer and the DanaBot banking malware. You can find a more in-depth analysis of the recent macros used by this actor in a post written by 0verfl0w [4].

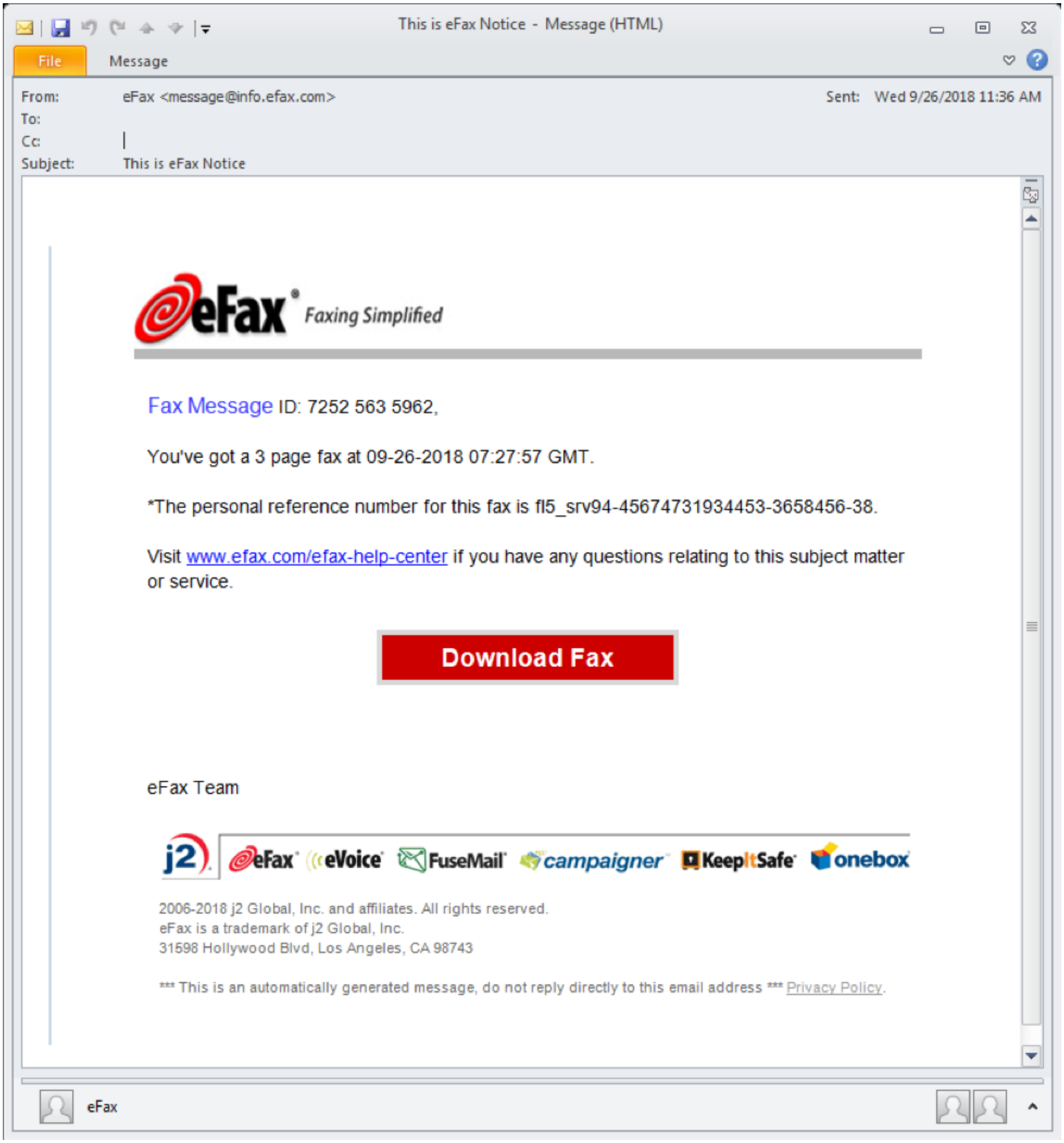


Figure 1: Message example with URLs linking to the download of a document containing macros that download the Hancitor payload

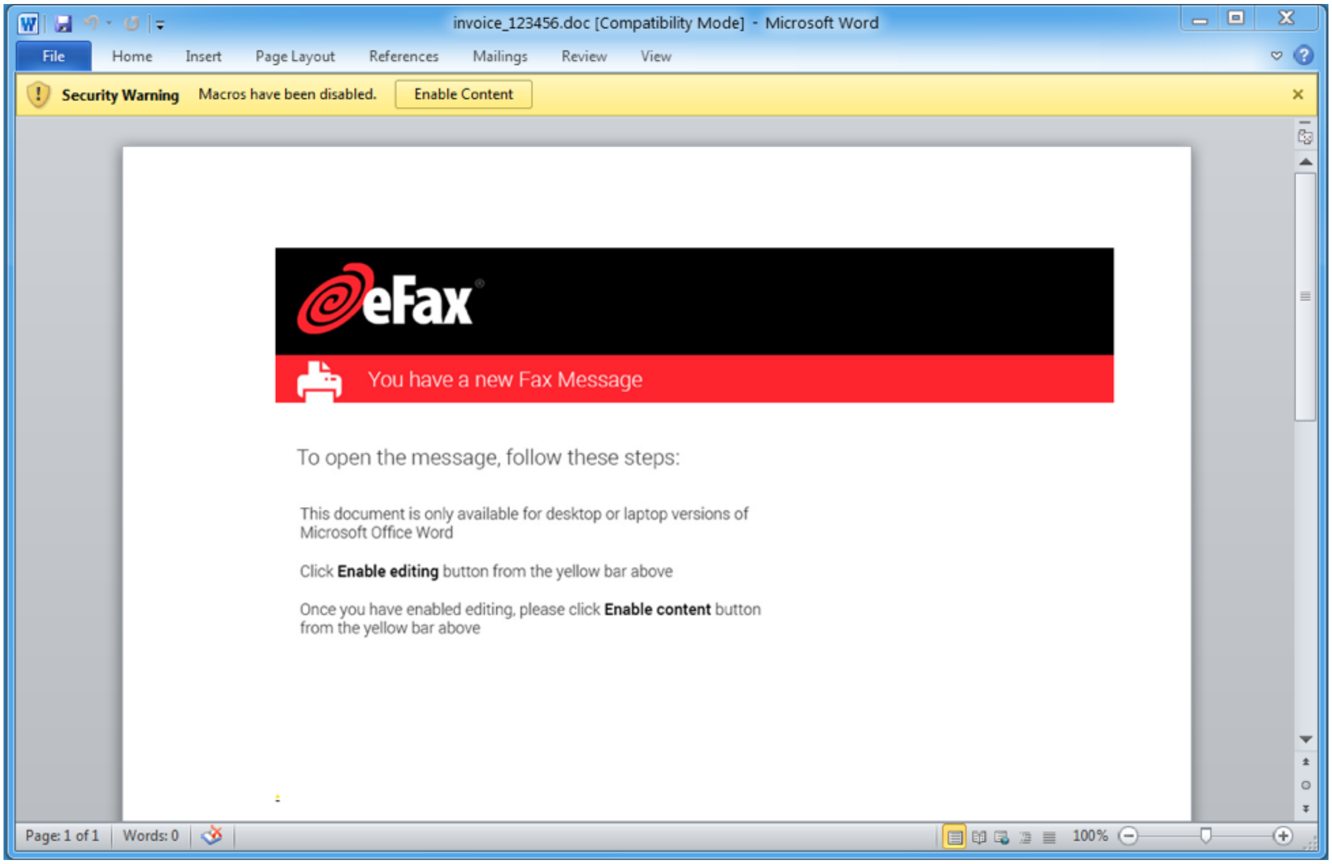


Figure 2: Macro document that contains the Hancitor payload

Malware Analysis (v2.003)

As previously described, DanaBot is a banking malware written in the Delphi programming language. This section continues our analysis of DanaBot by examining details of version 2.003. This is the latest version that we have seen in the wild, first appearing in early September. The version number is based on a version string (Figure 3) that is sometimes transmitted when the malware sends data to the command and control (C&C) server.

```
System Info
User: [redacted]
OS: Windows 7 Service Pack 1 (Version 6.1, Build 7601, 64-bit Edition)
Computer: [redacted]
Country: United States
Language: English
Time: [redacted] PM
WinKey: [redacted]
Desktop: 1024x768x24
Uptime: 0d[redacted]
HDDs: C(0mb/0mb)
Processes:
224=C:\Windows\System32\smss.exe
...
Version: 2.003 - x32
```

Figure 3: DanaBot's version string being sent to the C&C server along with system information

DanaBot is composed of three components:

1. Loader: downloads and loads main component
2. Main component: downloads, configures, and loads modules
3. Modules: various malware functionality

Anti-analysis

DanaBot includes a significant amount of junk code including extra instructions, conditional statements, and loops. When combined with the use of Delphi, these features dramatically impair reverse engineering. In addition, DanaBot uses Windows API function hashing and encrypted strings to prevent analysts and automated tools from easily determining the code's purpose.

A version of the API hashing algorithm written in Python [7], a list of the resolved Windows API functions used in the loader [8] and the main component [9] are available on Github.

The characters of the encrypted strings are stored as an array of DWORDs and are decrypted using a key and a basic substitution cipher. An IDA Pro Python script [10] and a list of decrypted strings used in the loader [11] and the main [12] component are available on Github.

Command & Control IPs

In both the loader and main components there is a list of 10 C&C IP addresses stored as DWORDs. Figure 4 shows an example from a memory dump of a loader component:

```
.data:004D2520 g_c2_45_77_231_138 dd 8AE74D2Dh          ; DATA XREF: XXX_path_to_comms:loc_4C7FDB↑r
.data:004D2520                                          ; Python>socket.inet_ntoa(struct.pack("I", 0x8AE74D2D))
.data:004D2520                                          ; 45.77.231.138
.data:004D2524 g_c2_149_154_152_64 dd 40989A95h         ; DATA XREF: XXX_path_to_comms+DF7↑r
.data:004D2528 g_c2_91_210_222_49 dd 31DED25Bh          ; DATA XREF: XXX_path_to_comms+EA5↑r
.data:004D252C g_c2_81_39_236_104 dd 68EC2751h          ; DATA XREF: XXX_path_to_comms+ECC↑r
.data:004D2530 g_c2_133_117_64_199 dd 0C7407585h        ; DATA XREF: XXX_path_to_comms+FFF↑r
.data:004D2534 g_c2_87_229_30_154 dd 9A1EE557h          ; DATA XREF: XXX_path_to_comms+1023↑r
.data:004D2538 g_c2_6_43_184_18 dd 12B82B06h            ; DATA XREF: XXX_path_to_comms:loc_4C82A9↑r
.data:004D253C g_c2_107_202_186_201 dd 0C9BACA6Bh       ; DATA XREF: XXX_path_to_comms:loc_4C8300↑r
.data:004D2540 g_c2_107_84_178_1 dd 1B2546Bh            ; DATA XREF: XXX_path_to_comms+11E8↑r
.data:004D2544 g_c2_216_45_35_66 dd 42232DD8h           ; DATA XREF: XXX_path_to_comms:loc_4C7F48↑r
.data:004D2548 ; char g_rsa_key[148]
.data:004D2548 g_rsa_key       db 6                     ; DATA XREF: decode_rsa_key_and_get_f39_data+40↑w
.data:004D2548                                          ; main_component_phonehome+C37↑o
.data:004D2549                 db    2
.data:004D254A                 db    0
.data:004D254B                 db    0
.data:004D254C                 db    0
.data:004D254D                 db 0A4h ; ¤
.data:004D254E                 db    0
.data:004D254F                 db    0
.data:004D2550                 db  52h ; R
.data:004D2551                 db  53h ; S
.data:004D2552                 db  41h ; A
.data:004D2553                 db  31h ; 1
.data:004D2554                 db    0
```

Figure 4: Example of C&C IP addresses in a memory dump of DanaBot's loader component

Note: Please see the "C&C Infrastructure" section for a potential caveat about these hard-coded IP addresses.

C&C Communications

In the previous versions we analyzed, DanaBot's loader component used HTTP for communications and its main component used a binary protocol. In version 2.003, both components use a binary protocol over TCP port 443. Despite the port number, it does not use TLS.

The protocol has some quirks, but in general consists of a 183-byte header followed by optional payload data. Most of the header values in a request are echoed back in the response header. If there is payload data, the format depends on the particular command.

Binary Protocol Header

An example of the header is shown in Figure 5.

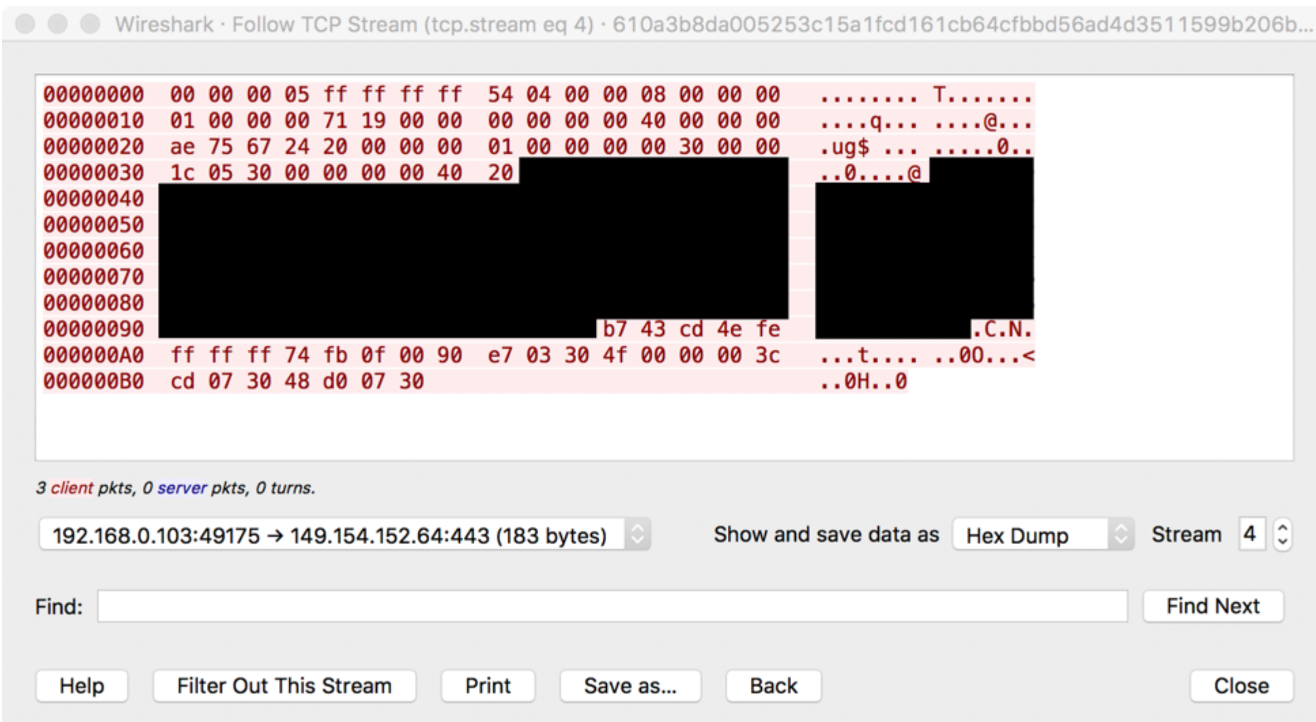


Figure 5: Example 183-byte header used in DanaBot's binary protocol

It can be broken down into the following fields:

- Offset 0: random values (stack junk) (DWORD)
- Offset 4: hardcoded -1 (DWORD)
- Offset 8: command (DWORD)
- Offset 0xc: affiliate ID (DWORD)
- Offset 0x10: hardcoded 1 (DWORD)
- Offset 0x14: random value based on a linear congruential generator (DWORD)
- Offset 0x18: unknown counter variable (DWORD)
- Offset 0x1c: system architecture (DWORD)
- Offset 0x20: Windows version information (DWORD)
- Offset 0x24: command argument (DWORD)
- Offset 0x28: admin status (DWORD)
- Offset 0x2c: process integrity level (DWORD)
- Offset 0x30: payload length (QWORD)
    - Depending on the command, this can contain random values (stack junk) instead

- Offset 0x38: length of next field (BYTE)
- Offset 0x39: bot ID (32 bytes)
  MD5 hex digest of various system information
- Offset 0x59: length of next field (BYTE)
- Offset 0x5a: command-dependent (32 bytes)
  - Can be used as part of an encryption key; in this case, it would be the MD5 hex digest of the bot ID (offset 0x39)
  - Can be used as a module identifier when requesting a module
- Offset 0x7a: length of next field (BYTE)
- Offset 0x7b: a nonce (32 bytes)
- Offset 0x9b - end of header: random values (stack junk)

Commands

We have identified and analyzed the following commands. The first command is performed by the loader, while the rest are performed by the main component.

Command 0x454 (1108): "Request main component"

This command is used by the loader to request the main component from the C&C server. The command argument (offset 0x24 in the header) will contain the integer "32" or "64" to request either the x86 version or x64 version of the component. The response payload contains encrypted data and an encrypted 128-byte RSA signature block used to verify the data. A decryption key is generated by the CryptDeriveKey Windows API function where it is initialized by taking the MD5 digest of the value at offset 0x5a in the header. Data is AES-256-CBC-encrypted using an initialization vector (IV) of 16 null (\x00) bytes. The decrypted data is the main component DLL which will be executed by rundll32.exe.

Command 0x453 (1107): "Initial beacon"

This is the first command sent by the main component to the C&C server. There is no data in the request or the response, so we believe this is just an initial beacon.

Command 0x44c (1100): "Request module identifiers"

This command is used by the malware to request a list of module identifiers from the C&C server. Figure 6 shows an example response listing these 6 module identifiers:

- 759CBB3E1B883BDCA23E9052462F641E
- E0FBBC92DB9927BFC474A64DF4F9C22F
- D0C851FBCA030928B535FAF3188DAFBA
- A5BBBAB3A17BA2119F47F0E4316EE5BF
- 4F06D71C93E4105307339704D21C49A3
- 8C59B6C9985F983E248E27CC0BF98A2D

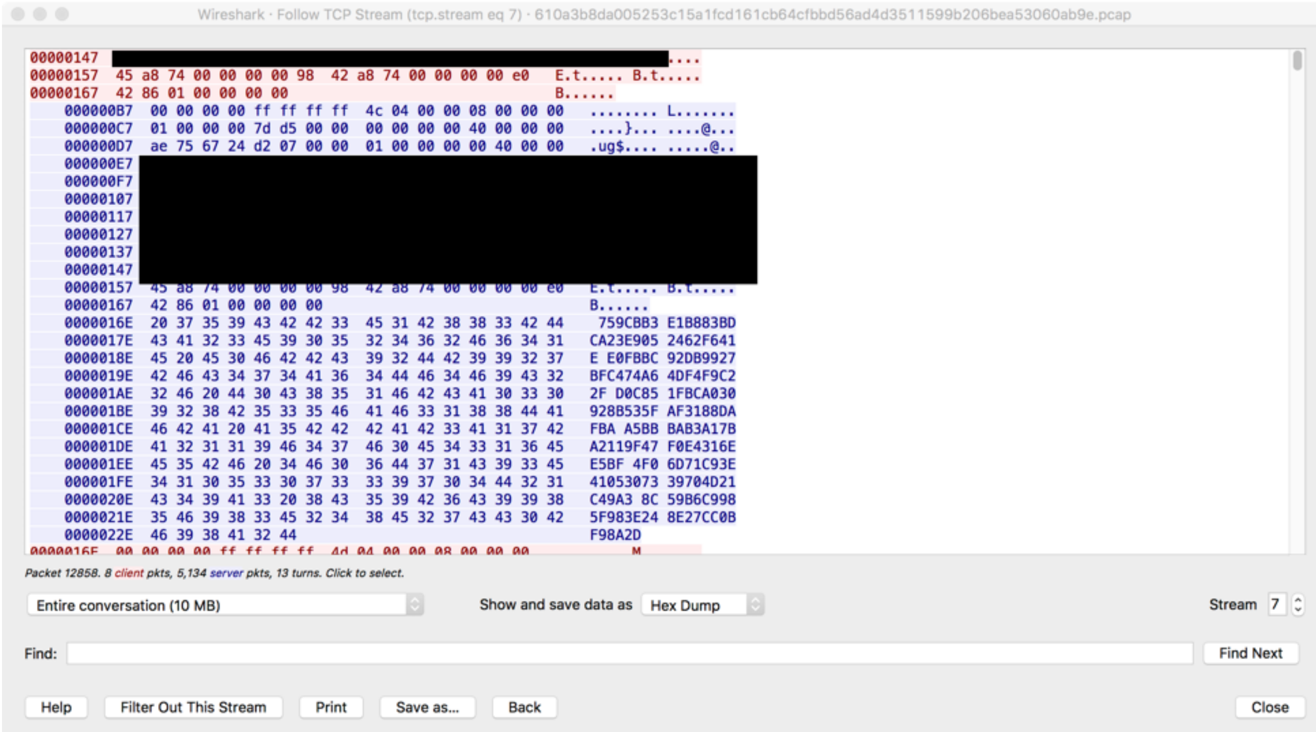


Figure 6: Command 0x44c response payload data containing a list of module identifiers

Command 0x44d (1101): “Request module”

This command is used to request a module from the C&C server. To indicate what module to download, field at offset 0x5a in the header will contain a module identifier (received via command 0x44c). The response payload data will contain a 1699-byte subheader, encrypted data, and a encrypted 128-byte RSA signature block used to verify the data. Figure 7 shows an example subheader:

```
00000000  93 f7 36 00 00 00 00 00  ff ff ff ff 01 00 00 00  |..6.............|
00000010  46 00 46 00 31 00 00 00  00 00 00 00 00 00 00 00  |F.F.1...........|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000210  00 00 00 00 00 00 00 00  46 00 46 00 31 00 2e 00  |........F.F.1...|
00000220  64 00 61 00 74 00 00 00  00 00 00 00 00 00 00 00  |d.a.t...........|
00000230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000640  00 00 00 00 00 00 00 00  00 20 37 35 39 43 42 42  |......... 759CBB|
00000650  33 45 31 42 38 38 33 42  44 43 41 32 33 45 39 30  |3E1B883BDCA23E90|
00000660  35 32 34 36 32 46 36 34  31 45 02 2d 2d 00 00 00  |52462F641E.--...|
00000670  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000680  00 00 20 00 00 00 05 00  00 00 01 00 00 00 01 00  |.. .............|
00000690  00 00 f0 f0 36 00 00 00  00 00 01 7d 97 a1 f3 f6  |....6......}....|
000006a0  29 e5 40                                          |).@|
000006a3
```

Figure 7: Command 0x44d response payload data containing 1699-byte subheader

The following fields have been identified in this subheader:

- Offset 0: total length of subheader and data (QWORD)
- Offset 8: hardcoded -1 (DWORD)

- Offset 0x10: module name (520-byte wide string)
- Offset 0x218: module filename (520-byte wide string)
- Offset 0x649: length of next field (BYTE)
- Offset 0x64a: module identifier (32 bytes)
- Offset 0x682: module architecture (DWORD)
- Offset 0x686: module type (DWORD)
- Offset 0x68e: data is ZLIB-compressed flag (DWORD)
- Offset 0x692: length of encrypted data (QWORD)

A decryption key (used to decrypt the module) is generated by the CryptDeriveKey Windows API function where it is initialized by the following process:

1. Copying the 1699-byte subheader into a buffer and zeroing the following fields:
    1. Offset 0: total length of subheader and data (QWORD)
    2. Offset 0x692: length of encrypted data (QWORD)
2. The buffer is MD5 hashed
3. The uppercase hex digest of the hash is itself MD5 hashed

Data is AES-256-CBC-encrypted using an initialization vector (IV) of 16 null (\x00) bytes. The decrypted data is optionally ZLIB compressed and once decompressed contains a module DLL that will be executed by rundll32.exe

Table 1: List of modules typically seen

Module identifier	**Name**	**Old name**	**Functionality**
759CBB3E1B883BDCA23E9052462F641E	FF1	Sniffer	Proxy
E0FBBC92DB9927BFC474A64DF4F9C22F	FF2	Stealer	Stealer module
D0C851FBCA030928B535FAF3188DAFBA	FF3	NA	64-bit version of Stealer module (new)
8C59B6C9985F983E248E27CC0BF98A2D	FF4	NA	RDP module (new)
A5BBBAB3A17BA2119F47F0E4316EE5BF	FF5	TOR	TOR proxy
4F06D71C93E4105307339704D21C49A3	FF6	VNC	VNC

Command 0x44f (1103): “Get configuration files”

This command is used by the malware to request configuration files from the C&C. It has a quirk where after the malware receives the 183-byte response header, the malware sends “\xff\xff\xff\xff\xff\xff\xff\xff” before the C&C server responds with the response payload data. The payload data is formatted and encrypted like a module, but multiple configuration files are sent (multiple 1699-byte subheader, encrypted data, and signature packages).

Table 2: Configuration files typically seen

Config filename	Variants	Purpose	Comments
BitVideo	VVie	Processes to watch	For screenshots/video recording perhaps
KeyBit	BitKey, VKey	Processes to watch	For keylogging possibly
BitFiles	Vfiles, VBit	Cryptocurrency wallet files to steal	
PosWtFilter	PostWFilter, VFilter	List of websites for which to steal requests	PosWtFilter may be a typo (in affiliate IDs 3 and 9)
webinj33	uabanks	Proxying config	Incrementing versions
inj25	InjectZZ, InjectSW	Webinjects	Incrementing versions; Zeus-style injects

This command is used by the malware to send data to the C&C such as the system information (Figure 3 above) or a screenshot. The request payload data contains a 656-byte subheader, encrypted data, and encrypted session key (Figure 8 shows an example subheader):Command 0x44e (1102): “Send data to C&C”

```
00000000  14 72 08 00 00 00 00 00  ff ff ff ff 08 00 00 00  |.r..............|
00000010  00 00 00 00 00 00 00 20  [redacted]
00000020  [redacted]
00000030  [redacted]
00000040  [redacted]
00000050  [redacted]  45 00 64 00 65 00 73 00  |[redacted].d.e.s.|
00000060  6b 00 74 00 6f 00 70 00  73 00 63 00 72 00 65 00  |k.t.o.p.s.c.r.e.|
00000070  65 00 6e 00 2e 00 62 00  6d 00 70 00 00 00 00 00  |e.n...b.m.p.....|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000260  00 00 00 00 64 00 00 00  02 00 00 00 01 00 00 00  |....d...........|
00000270  [redacted]  00 00 00 00 00 00 00 00  |1...x,.@........|
00000280  98 fe ff ff 00 00 00 00  84 6f 08 00 00 00 00 00  |.........o......|
00000290
```

Figure 8: Command 0x44e request payload data containing 656-byte subheader

The following fields have been identified in the subheader:

- Offset 0: total length (QWORD)
- Offset 8: hardcoded -1 (DWORD)
- Offset 0xc: affiliate ID (DWORD)
- Offset 0x17: length of next field (BYTE)
- Offset 0x18: bot ID (32 bytes)
- Offset 0x38: length of next field (BYTE)
- Offset 0x39: MD5 hex digest of plaintext data (32 bytes)
- Offset 0x5a: filename (520-byte wide string)
- Offset 0x264: data type (DWORD)
- Offset 0x270: system time (unknown format) (QWORD)
- Offset 0x280: timezone bias (DWORD)
- Offset 0x288: encrypted data length (QWORD)

Data can be ZLIB-compressed and AES-256-CBC-encrypted using an initialization vector (IV) of 16 null (\x00) bytes. The encryption key is generated using the CryptDeriveKey Windows function and RSA-encrypted using an embedded RSA public key. The RSA-encrypted AES key is then appended to the end of the encrypted data.

Table 3: Files typically seen sent

Filename	Comments
desktopscreen.bmp	Screenshot
Cookies.txt	Stored web browser cookies
“System Info"	Various system information

C&C Infrastructure

While we do not have specific visibility into DanaBot’s back-end infrastructure, we have observed some noteworthy behavior that allows some speculation.

As noted above, DanaBot uses a loader to download its main component from a C&C server. The main component contains a list of 10 hardcoded C&C IP addresses that are used for malware communications. Our first observation was that the hardcoded C&C lists changed approximately every hour when a main component was downloaded. We downloaded the main component in hourly intervals for 24 hours and analyzed the C&C lists. Each sample’s list turned out to be different. Overall we ended up with 240 IP addresses (available on Github [13]) with 194 (80%) of them being unique. The top 10 overlapping IPs were:

- 158.255.215[.]31 (in 7 lists)
- 149.154.152[.]64 (in 7 lists)
- 37.235.53[.]232 (in 6 lists)
- 95.179.151[.]252 (in 5 lists)
- 178.209.51[.]227 (in 5 lists)
- 149.154.157[.]220 (in 5 lists)

- 45.77.54[.]180 (in 4 lists)
- 45.77.96[.]198 (in 3 lists)
- 45.77.51[.]69 (in 3 lists)
- 45.77.231[.]138 (in 3 lists)

Out of the total list of possible C&C IPs, only the following 10 (4%) seemed responsive:

- 149.154.152[.]64
- 149.154.157[.]220
- 158.255.215[.]31
- 178.209.51[.]227
- 37.235.53[.]232
- 45.77.231[.]138
- 45.77.51[.]69
- 45.77.54[.]180
- 45.77.96[.]198
- 95.179.151[.]252

Interestingly, these synced up with the overlapping IP list. We also noted that the overall IP list contained some unrouteable IPs such as:

- 10.181.255[.]78
- 225.100.146[.]224
- 225.21.55[.]173
- 226.181.243[.]104
- 228.226.171[.]37
- 234.106.187[.]114
- 234.63.249[.]87
- 234.97.12[.]178
- 235.40.105[.]171
- 238.87.111[.]55

As a result of these observations, we can speculate that the main component may contain only a few real C&Cs while the rest are random decoys.

Affiliate System

Based on distribution methods and targeting, we have been grouping DanaBot activity using an "affiliate ID" that we have observed in various part of the C&C protocol (e.g., offset 0xc of the 183-byte binary protocol header). At the time of publication, we observed the following affiliate IDs:

Affiliate ID	Targeting	Distribution
3	Poland, Austria, Germany, Italy	Zipped-VBS attachments in email campaigns
4	Australia	Links in email campaigns

5	No webinjects	unknown
8	UK, Ukraine, and Canada	Various email campaigns
9	Same as affiliate ID 3	Fallout Exploit Kit
11	US, No webinjects	Hancitor downloader malware from links in email campaigns
12	Australia	unknown
13	Germany	unknown
20	No webinjects	unknown

We observed that DanaBot samples with different affiliate IDs seem to use some of the same C&C IP addresses. At this point we speculate that DanaBot may be set up as a “malware as a service” system in which one threat actor controls a global C&C panel and infrastructure system and then sells access to other threat actors (affiliates) who distribute and target DanaBot as they see fit.

Comparison with CryptXXX Ransomware

Proofpoint blogged about CryptXXX file-encrypting ransomware in 2016 [5] and noted that it shared many similarities with Reveton “police” ransomware. In particular, we noted that it was written in Delphi and used a custom command and control protocol on TCP port 443.

DanaBot’s C&C traffic appears to be an evolution of this protocol, now using AES encryption in addition to the Zlib compression. For example, in the traffic included in the Malware Traffic Analysis blog [6], the initial CryptXXX checkin format is:

```
00000000  20 35 34 37 43 34 36 46  35 41 43 38 38 34 36 34  | 547C46F5AC88464|
00000010  36 45 35 45 33 46 36 44  38 31 36 33 42 33 30 42  |6E5E3F6D8163B30B|
00000020  38 00 00 00 91 70 00 00  00 00 00 00 00 00 00 00  |8....p..........|
00000030  e8 03 00 00 4e 00 00 00  78 01 fb fb ff ff 7f 05  |....N...x.......|
00000040  53 13 73 67 13 33 37 53  47 67 0b 0b 13 33 13 33  |S.sg.37SGg...3.3|
00000050  57 53 57 63 37 33 17 0b  43 33 63 27 63 03 27 0b  |WSWc73..C3c'c.'.|
00000060  86 a1 02 d8 43 0d 40 c0  92 38 f7 b2 1a ea 19 18  |....C.@..8......|
00000070  18 12 a7 16 a4 ca 16 88  1d 40 0c 1c e0 05 33 03  |.........@....3.|
00000080  03 00 5f 2a 0f 30
```

Figure 9: CryptXXX checkin format

The following fields are among those common to both CryptXXX and DanaBot:

- Offset 0: length of next field (BYTE)
- Offset 2: bot ID (32 bytes)

- Offset 0x34 : length of compressed buffer
- Offset 0x38: Zlib-compressed buffer (0x4e bytes)

The compressed buffer decodes to:

```
00000000  fd ff ff ff 20 35 34 37  43 34 36 46 35 41 43 38  |.... 547C46F5AC8|
00000010  38 34 36 34 36 45 35 45  33 46 36 44 38 31 36 33  |84646E5E3F6D8163|
00000020  42 33 30 42 38 00 00 00  00 00 00 00 00 00 00 00  |B30B8...........|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 07 55  |...............U|
000000d0  30 30 30 30 30 39 00 00  00 00 00 00 00 00 00 00  |000009..........|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 05 31 2e 30  |.............1.0|
00000100  30 31 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |01..............|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 3d 00 00 00  |............=...|
00000130  40 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |@...............|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 e8 03 00 00  |................|
00000150
```

Figure 10: Decoded payload buffer

The following fields have been identified in the decoded buffer:

- Offset 4: length of next field (BYTE)
- Offset 5: bot ID (32 bytes)
- Offset 0xce : length of next field (BYTE)
- Offset 0xcf : Affiliate ID (7 bytes)
- Offset 0xfc : length of next field (BYTE)
- Offset 0xfd : Version string (5 bytes)
- etc

Later on in the communication there is a (decoded) request to download a “Stealer” module “stiller.dll”:

```
00000000  0b 00 00 00 40 41 75 38  44 44 4b 33 7a 34 5a 30  |....@Au8DDK3z4Z0|
00000010  41 39 62 38 63 46 65 46  46 38 47 46 68 30 71 45  |A9b8cFeFF8GFh0qE|
00000020  71 37 45 41 72 46 74 43  55 39 69 34 61 30 73 41  |q7EArFtCU9i4a0sA|
00000030  64 30 4d 34 4c 38 5a 41  50 37 41 41 62 44 43 38  |d0M4L8ZAP7AAbDC8|
00000040  64 34 46 33 46 00 00 00  00 00 00 00 00 00 00 00  |d4F3F...........|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 0b 73 74 69 6c 6c 65  |..........stille|
00000070  72 2e 64 6c 6c 67 37 6b  7a 69 2e 6f 6e 69 6f 6e  |r.dllg7kzi.onion|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figure 11: Decoded request to download the “Stealer” module

Thus it would seem that Danabot follows in a long line of malware from one particular group. This family began with ransomware, to which stealer functionality was added in Reveton. The evolution continued with CryptXXX ransomware and now with a banking Trojan with Stealer and remote access functionality added in Danabot.

Conclusion

When we first discovered DanaBot, we predicted that it would likely be picked up by other actors. Distribution of this malware has now extended well beyond Australia, with campaigns targeting Poland, Italy, Germany, Austria, and, more recently, the United States. DanaBot is a banking Trojan, meaning that it is necessarily geo-targeted to a degree. Adoption by high-volume actors, though, as we saw in the US campaign, suggests active development, geographic expansion, and ongoing threat actor interest in the malware. The malware itself contains a number of anti-analysis features, as well as updated stealer and remote control modules, further increasing its attractiveness and utility to threat actors.

References

[1] https://www.proofpoint.com/us/threat-insight/post/danabot-new-banking-trojan-surfaces-down-under-0

[2] https://www.welivesecurity.com/2018/09/21/danabot-targeting-europe-adds-new-features/

[3] https://www.proofpoint.com/us/threat-insight/post/hancitor-ruckguv-reappear

[4] https://0ffset.wordpress.com/2018/08/12/post-0x16-hancitor-stage-1/

[5] https://www.proofpoint.com/us/threat-insight/post/cryptxxx-new-ransomware-actors-behind-reveton-dropping-angler

[6] http://malware-traffic-analysis.net/2016/04/20/index.html

[7] https://github.com/EmergingThreats/threatresearch/blob/master/danabot/func_hashes.py

[8] https://github.com/EmergingThreats/threatresearch/blob/master/danabot/loader_func_hashes.txt

[9] https://github.com/EmergingThreats/threatresearch/blob/master/danabot/main_func_hashes.txt

[10] https://github.com/EmergingThreats/threatresearch/blob/master/danabot/decrypt_str_ida.py

[11] https://github.com/EmergingThreats/threatresearch/blob/master/danabot/loader_strings.txt

[12] https://github.com/EmergingThreats/threatresearch/blob/master/danabot/main_strings.txt

[13] https://github.com/EmergingThreats/threatresearch/blob/master/danabot/24_hours_of_ips.txt

Indicators of Compromise (IOCs)

IOC	IOC Type	Description
288615e28672e1326231186230f2bc74ea84191745cc40369d49bf385bf9669b	SHA256	DanaBot Loader (affiliate ID 8)

45.77.96.198	IP Address	DanaBot Loader C&C
57cac2bdc44415c6737149bda8fc4e53adfab7d35cac3de94ced9d6675f1c5db	SHA256	DanaBot Main x64 (affiliate ID 8)
1184c7936c82f1718f9e547be4a8eeaa1c16c2f16790e2b5ae66a870a17b7454	SHA256	DanaBot Main x86 (affiliate ID 8)
149.154.152.64	IP Address	DanaBot Main C&C
149.154.157.220	IP Address	DanaBot Main C&C
158.255.215.31	IP Address	DanaBot Main C&C
178.209.51.227	IP Address	DanaBot Main C&C
37.235.53.232	IP Address	DanaBot Main C&C
45.77.231.138	IP Address	DanaBot Main C&C
45.77.51.69	IP Address	DanaBot Main C&C
45.77.54.180	IP Address	DanaBot Main C&C
45.77.96.198	IP Address	DanaBot Main C&C

Hancitor Campaign IOCs:

genesislouisville[.]com	Domain	Link to macro document
genesisofdallas[.]com	Domain	Link to macro document
genesisoflouisville[.]com	Domain	Link to macro document
genesisofportland[.]com	Domain	Link to macro document
kccmanufacturing[.]com	Domain	Link to macro document
louisvillegenesis[.]com	Domain	Link to macro document
louisvilleride[.]com	Domain	Link to macro document
motionscent[.]com	Domain	Link to macro document
oxmoorautomall[.]com	Domain	Link to macro document
ridesharelouisville[.]com	Domain	Link to macro document
6dcf41dd62e909876e9ef10bd376ea3a6765c2ecb281844fc4bebd70bfebeb27	SHA256	Macro document
c82081823ba468ad2d10c4beca700a7bf0ba82b371bc57286cc721e271019080	SHA256	Hancitor

hxxp://tontheckcatan[.]ru/4/forum[.]php	URL	Hancitor C&C
hxxp://onthethatsed[.]ru/4/forum[.]php	URL	Hancitor C&C
hxxp://kitezona[.]ru/wp-content/plugins/redirection/modules/1	URL	Hancitor Task
hxxp://xn--hllo-bpa[.]com/guestlist/1	URL	Hancitor Task
hxxp://music-open[.]com/1	URL	Hancitor Task
hxxp://allnicolerichie[.]com/wp-content/plugins/ubh/1	URL	Hancitor Task
hxxp://mpressmedia[.]net/wp-content/plugins/ubh/1	URL	Hancitor Task
hxxp://bwc[.]jianbell[.]com/wp-content/plugins/ubh/1	URL	Hancitor Task
hxxp://kitezona[.]ru/wp-content/plugins/redirection/modules/2	URL	Hancitor Task
hxxp://xn--hllo-bpa[.]com/guestlist/2	URL	Hancitor Task
hxxp://music-open[.]com/2	URL	Hancitor Task
hxxp://allnicolerichie[.]com/wp-content/plugins/ubh/2	URL	Hancitor Task
hxxp://mpressmedia[.]net/wp-content/plugins/ubh/2	URL	Hancitor Task
hxxp://bwc[.]jianbell[.]com/wp-content/plugins/ubh/2	URL	Hancitor Task
hxxp://kitezona[.]ru/wp-content/plugins/redirection/modules/4	URL	Hancitor Task

hxxp://xn--hllo-bpa[.]com/guestlist/4	URL	Hancitor Task
hxxp://music-open[.]com/4	URL	Hancitor Task
hxxp://allnicolerichie[.]com/wp-content/plugins/ubh/4	URL	Hancitor Task
hxxp://mpressmedia[.]net/wp-content/plugins/ubh/4	URL	Hancitor Task
hxxp://bwc[.]ianbell[.]com/wp-content/plugins/ubh/4	URL	Hancitor Task
9a816d9626f870617400df384d653b02a15ad940701b4fb2296e1abe04d3777f	SHA256	DanaBot
hxxp://tontheckcatan[.]ru/mlu/forum[.]php	URL	Pony C&C
hxxp://onthethatsed[.]ru/mlu/forum[.]php	URL	Pony C&C
hxxp://tontheckcatan[.]ru/d2/about[.]php	URL	Pony C&C
hxxp://onthethatsed[.]ru/d2/about[.]php	URL	Pony C&C

ET and ETPRO Suricata/Snort Signatures

2819978 | ETPRO TROJAN Tordal/Hancitor/Chanitor Checkin

2014411 | ET TROJAN Fareit/Pony Downloader Checkin 2

2831891 | ETPRO CURRENT_EVENTS Hancitor Encrypted Payload Jul 19

2832816 | ETPRO TROJAN Win32/DanaBot CnC Checkin (affid 11)

Subscribe to the Proofpoint Blog