Dragon Threat Labs

June 28

2015

Insight in to advances of adversary tactics, techniques and procedures through analysis of an attack against an organisation in the Asia Pacific region.

Dragon Threat Labs, Hong Kong | Author: Dan

Contents

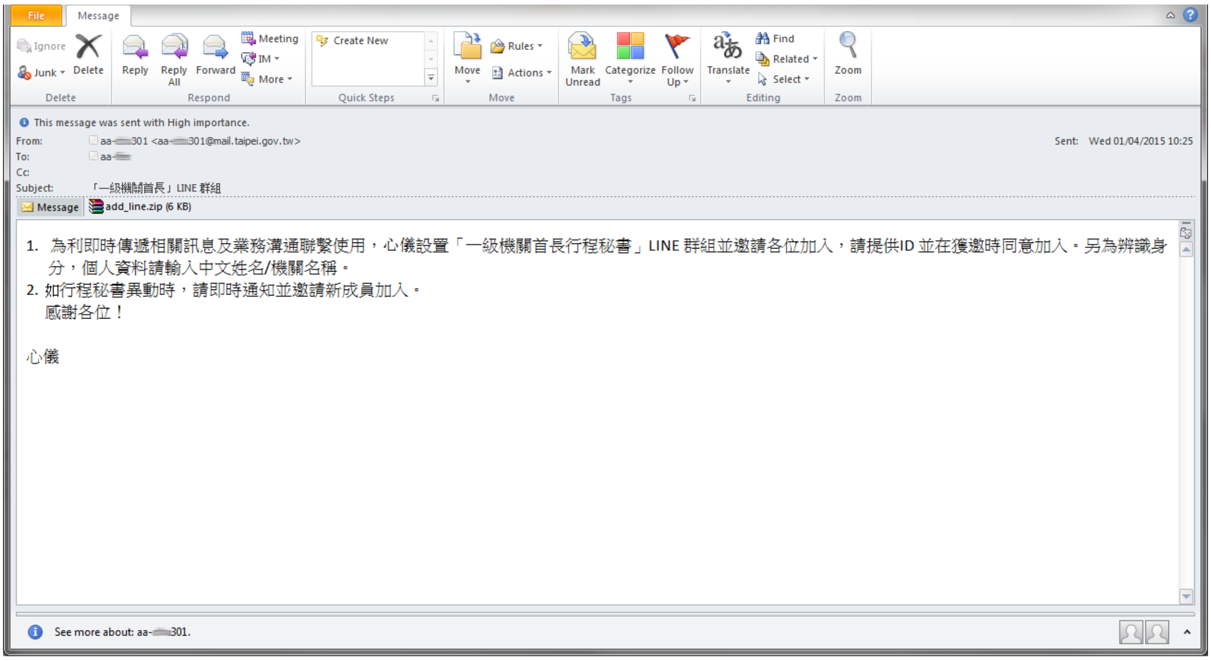
Introduction

The summer months dawn on us the financial year comes to a close. It is in the run up to this time that most organisations see an increase in targeted attack activity.

We begin by reading news of an attack against the Taiwanese Government. Whilst we would prefer to disassociate ourselves with APT attacks against Governments our interest was piqued by a particular blog written by our friends over at TrendMicro[1]. There were several things that struck us as both interesting and concerning about the details; a threat actor known to operate in South East Asia is now using secure sockets layer ("SSL") encryption in their malware. SSL is typically used to encrypt data between the client and the server, thus making the content unreadable by any systems sitting between the two end points, and significantly raising the cost of defence. Without the use of SSL interception traditional IDS/IPS systems will cease to detect compromised systems. For large organisations the cost of this can run from the hundreds of thousands in to the millions of dollars. For smaller organisations this single expense easily can run over the yearly security budget. Whilst a small, fun loving, not-for-profit group of misfits such as ourselves are not concerned with financial costs we are concerned with an adversary's change in behaviour, especially the use of encryption - have Snowden's actions[2] really affected the entire world?

Phishing with a hook and LINE

As most spearphishing stories begin, Mary receives an email from John, except the email isn't really from John, it's from somebody pretending to be John in an attempt to gain Mary's trust so that she'll open an email attachment that contains malware.



This message was sent with High importance.

From: aa-[redacted]301 <aa-[redacted]301@mail.taipei.gov.tw>

Sent: Wed 01/04/2015 10:25

To: aa-[redacted]

Cc:

Subject: 「一級機關首長」LINE 群組

Message add_line.zip (6 KB)

1. 為利即時傳遞相關訊息及業務溝通聯繫使用，心儀設置「一級機關首長行程秘書」LINE 群組並邀請各位加入，請提供ID 並在獲邀時同意加入。另為辨識身分，個人資料請輸入中文姓名/機關名稱。
2. 如行程秘書異動時，請即時通知並邀請新成員加入。
   感謝各位！

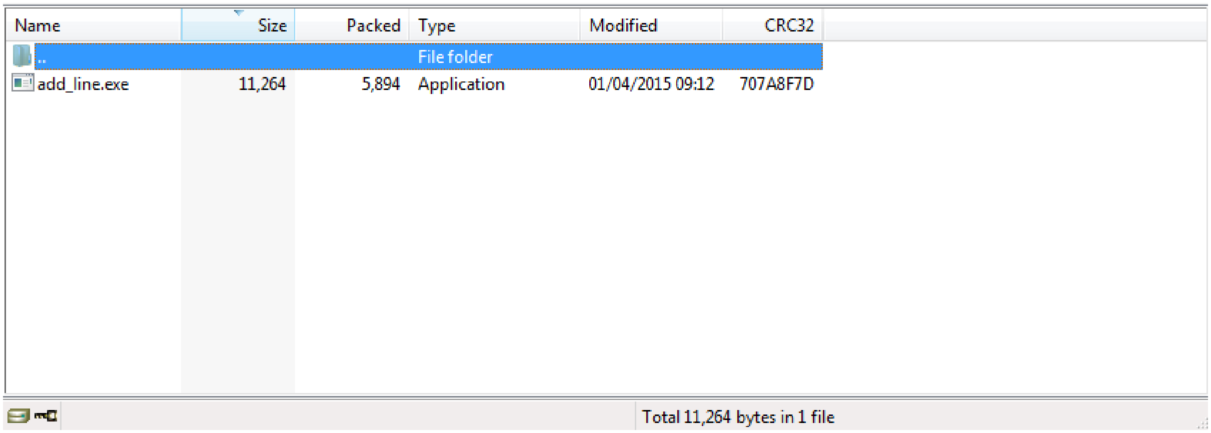
心儀

See more about: aa-[redacted]301.

[1] The blog posting has unfortunately since been removed

[2] Snowden's actions: we don't care what effect he had - he's still a prick. Welcome to Hong Kong ☺

In this case ‘Mary’ is an employee of the Taiwanese Government and ‘John’ is supposedly a co-worker also working for the Government. Based on the format of the email addresses it appears that the attacker has some working knowledge of their target organisation but the body of the email does not give away further ‘guilty knowledge’ and simply asks the user to open the attachment to install LINE, a popular instant messaging program used in millions of people in Taiwan.
The malware is very simply contained within a zip file. The zip file does not have a password. Fortunately in this case it seems that this email was noticed as suspicious by the recipient and they uploaded it to a popular anti-virus website.

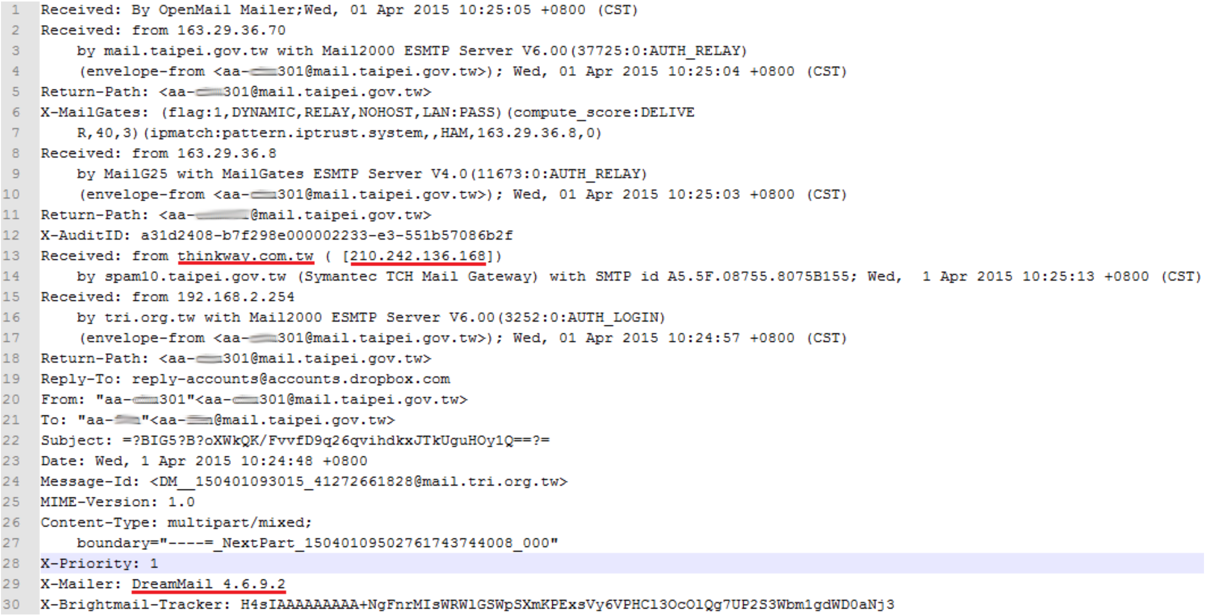


Name	Size	Packed	Type	Modified	CRC32
..			File folder		
add_line.exe	11,264	5,894	Application	01/04/2015 09:12	707A8F7D

Total 11,264 bytes in 1 file

This method of delivering malware isn’t uncommon in Asia. Due to a lack of general awareness in IT security many users fall victim to such attacks be it APT or common crimeware. It is of course good practice to block all executable files in email attachment (.exe, .bat, .cmd, .scr, .jar etc.). Whilst this method isn’t the most sophisticated don’t let it fool you – it proves to be very effective.

A further look into the email headers shows us that the email did not come from a co-worker; it in fact came from somebody outside of the organisation.



```
Received: By OpenMail Mailer;Wed, 01 Apr 2015 10:25:05 +0800 (CST)
Received: from 163.29.36.70
    by mail.taipei.gov.tw with Mail2000 ESMTP Server V6.00(37725:0:AUTH_RELAY)
    (envelope-from <aa-[redacted]301@mail.taipei.gov.tw>); Wed, 01 Apr 2015 10:25:04 +0800 (CST)
Return-Path: <aa-[redacted]301@mail.taipei.gov.tw>
X-MailGates: (flag:1,DYNAMIC,RELAY,NOHOST,LAN:PASS)(compute_score:DELIVE
    R,40,3)(ipmatch:pattern.iptrust.system,,HAM,163.29.36.8,0)
Received: from 163.29.36.8
    by MailG25 with MailGates ESMTP Server V4.0(11673:0:AUTH_RELAY)
    (envelope-from <aa-[redacted]301@mail.taipei.gov.tw>); Wed, 01 Apr 2015 10:25:03 +0800 (CST)
Return-Path: <aa-[redacted]@mail.taipei.gov.tw>
X-AuditID: a31d2408-b7f298e000002233-e3-551b57086b2f
Received: from thinkway.com.tw ( [210.242.136.168])
    by spam10.taipei.gov.tw (Symantec TCH Mail Gateway) with SMTP id A5.5F.08755.8075B155; Wed,  1 Apr 2015 10:25:13 +0800 (CST)
Received: from 192.168.2.254
    by tri.org.tw with Mail2000 ESMTP Server V6.00(3252:0:AUTH_LOGIN)
    (envelope-from <aa-[redacted]301@mail.taipei.gov.tw>); Wed, 01 Apr 2015 10:24:57 +0800 (CST)
Return-Path: <aa-[redacted]301@mail.taipei.gov.tw>
Reply-To: reply-accounts@accounts.dropbox.com
From: "aa-[redacted]301"<aa-[redacted]301@mail.taipei.gov.tw>
To: "aa-[redacted]"<aa-[redacted]@mail.taipei.gov.tw>
Subject: =?BIG5?B?oXWkQK/FvvfD9q26qvihdkxJTkUguHOy1Q==?=
Date: Wed, 1 Apr 2015 10:24:48 +0800
Message-Id: <DM__150401093015_41272661828@mail.tri.org.tw>
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="----=_NextPart_150401095027617437440O8_000"
X-Priority: 1
X-Mailer: DreamMail 4.6.9.2
X-Brightmail-Tracker: H4sIAAAAAAAAA+NgFnrMIsWRW1GSWpSXmKPExsVy6VPHCl3OcO1Qg7UP2S3Wbm1gdWD0aNj3
```

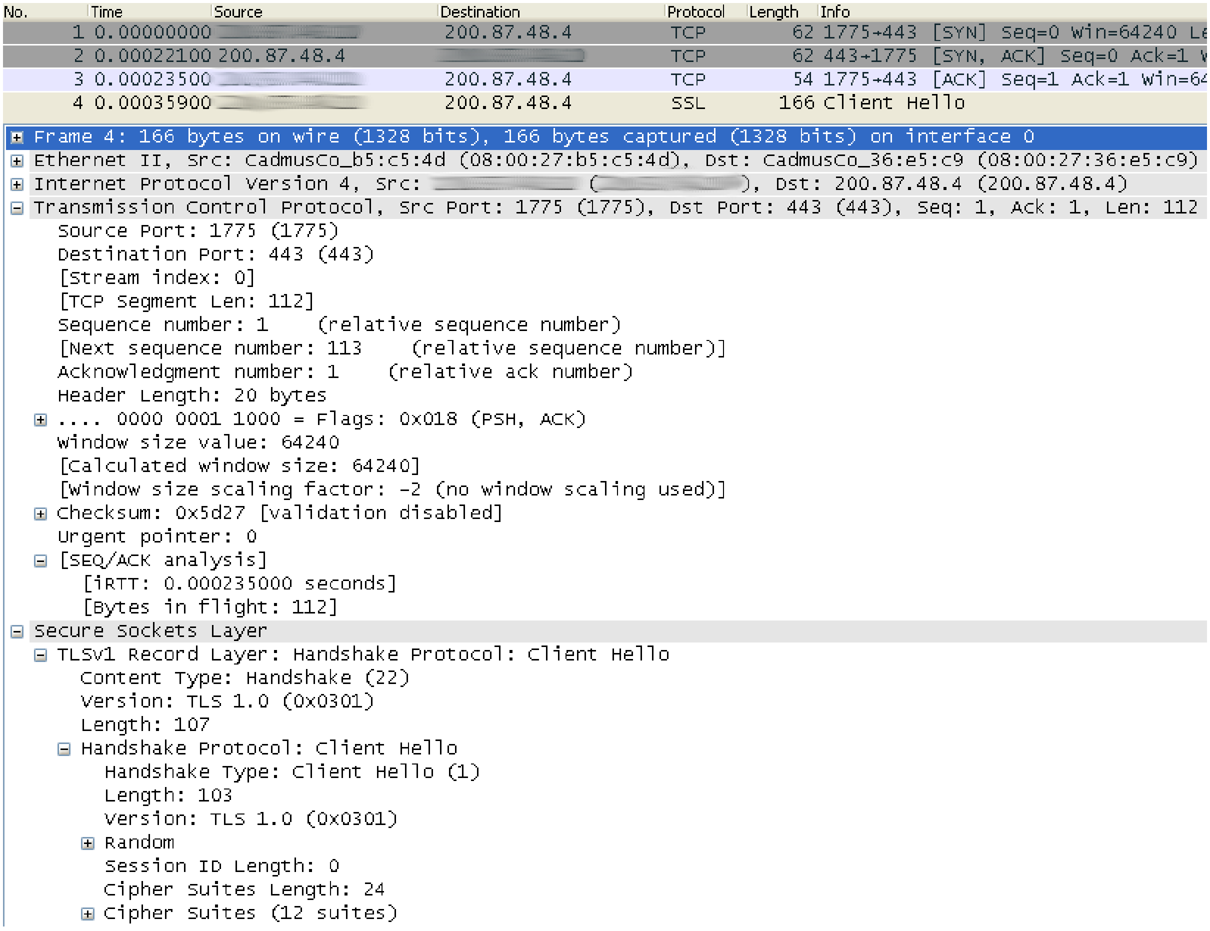
Many organisations worldwide complain of spearphishes coming from HiNet IP ranges and brandishing the DreamMail X-mailer. Unfortunately this combination of characteristics is very common in Asia and thus does not always make a good heuristic detection rule.

Mocelpa

Upon first look we notice that the executable file contained with the zip archive is fairly small being just 11 Kilobytes in size. Up until recently Mocelpa had a very low detection rate, being detected by just 1 out of 57 anti-virus engines tested against.

The first characteristic we notice is the lack of bootstrap mechanism. Should the user logout, shutdown or reboot the malware will not survive. This is interesting behaviour and suggests that the malware author is confident in Mocelpa's stability.

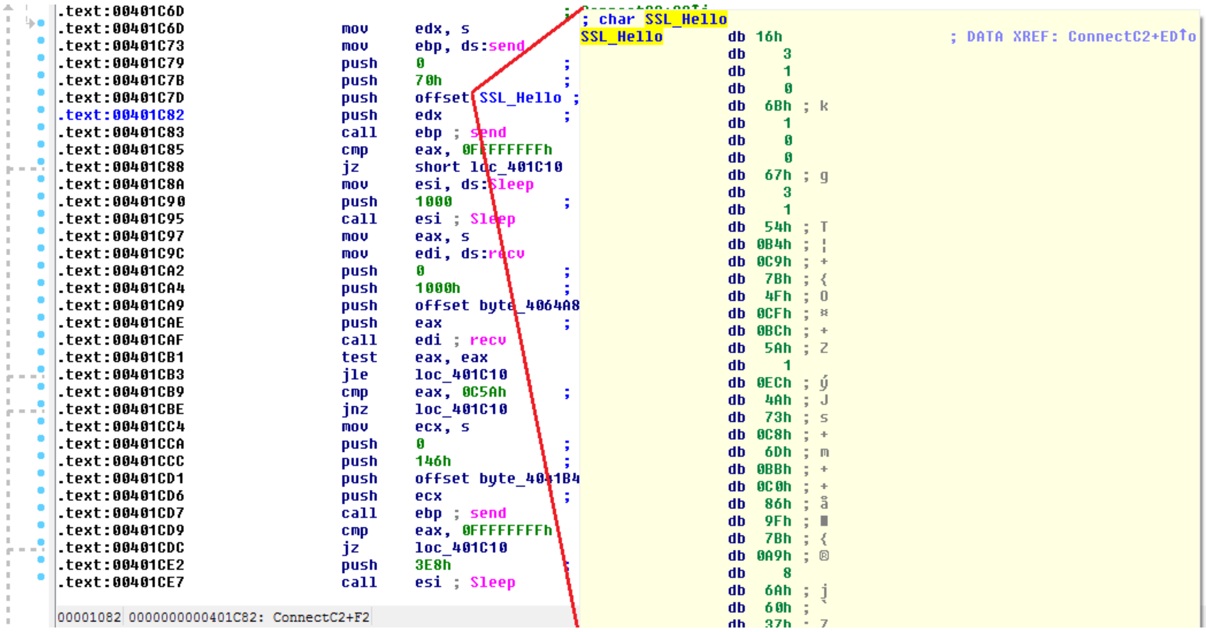
To begin with I will describe the network functions, since my main interest in this malware stems from the use of SSL. An initial glance at network traffic produced by Mocelpa reveals something interesting and surprising: an SSL handshake followed by quite blatant non-SSL traffic.



```
No.     Time        Source          Destination     Protocol  Length  Info
      1 0.00000000  [redacted]      200.87.48.4     TCP           62 1775→443 [SYN] Seq=0 Win=64240 Le
      2 0.00022100  200.87.48.4     [redacted]      TCP           62 443→1775 [SYN, ACK] Seq=0 Ack=1 W
      3 0.00023500  [redacted]      200.87.48.4     TCP           54 1775→443 [ACK] Seq=1 Ack=1 Win=64
      4 0.00035900  [redacted]      200.87.48.4     SSL          166 Client Hello

⊞ Frame 4: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface 0
⊞ Ethernet II, Src: CadmusCo_b5:c5:4d (08:00:27:b5:c5:4d), Dst: CadmusCo_36:e5:c9 (08:00:27:36:e5:c9)
⊞ Internet Protocol Version 4, Src: [redacted] ([redacted]), Dst: 200.87.48.4 (200.87.48.4)
⊟ Transmission Control Protocol, Src Port: 1775 (1775), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 112
    Source Port: 1775 (1775)
    Destination Port: 443 (443)
    [Stream index: 0]
    [TCP Segment Len: 112]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 113    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header Length: 20 bytes
  ⊞ .... 0000 0001 1000 = Flags: 0x018 (PSH, ACK)
    Window size value: 64240
    [Calculated window size: 64240]
    [Window size scaling factor: -2 (no window scaling used)]
  ⊞ Checksum: 0x5d27 [validation disabled]
    Urgent pointer: 0
  ⊟ [SEQ/ACK analysis]
      [iRTT: 0.000235000 seconds]
      [Bytes in flight: 112]
⊟ Secure Sockets Layer
  ⊟ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 107
    ⊟ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 103
        Version: TLS 1.0 (0x0301)
      ⊞ Random
        Session ID Length: 0
        Cipher Suites Length: 24
      ⊞ Cipher Suites (12 suites)
```

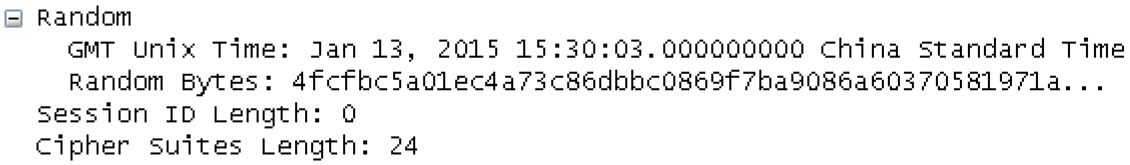
Delving into the disassembly behind the malware we can see that this SSL handshake is clearly faked and generated using hardcoded values within the malware.



```
.text:00401C6D
.text:00401C6D                 mov     edx, s
.text:00401C73                 mov     ebp, ds:send
.text:00401C79                 push    0                  ;
.text:00401C7B                 push    70h                ;
.text:00401C7D                 push    offset SSL_Hello ;
.text:00401C82                 push    edx                ;
.text:00401C83                 call    ebp ; send
.text:00401C85                 cmp     eax, 0FFFFFFFFh
.text:00401C88                 jz      short loc_401C10
.text:00401C8A                 mov     esi, ds:Sleep
.text:00401C90                 push    1000               ;
.text:00401C95                 call    esi ; Sleep
.text:00401C97                 mov     eax, s
.text:00401C9C                 mov     edi, ds:recv
.text:00401CA2                 push    0                  ;
.text:00401CA4                 push    1000h              ;
.text:00401CA9                 push    offset byte_4064A8
.text:00401CAE                 push    eax                ;
.text:00401CAF                 call    edi ; recv
.text:00401CB1                 test    eax, eax
.text:00401CB3                 jle     loc_401C10
.text:00401CB9                 cmp     eax, 0C5Ah         ;
.text:00401CBE                 jnz     loc_401C10
.text:00401CC4                 mov     ecx, s
.text:00401CCA                 push    0                  ;
.text:00401CCC                 push    146h               ;
.text:00401CD1                 push    offset byte_4041B4
.text:00401CD6                 push    ecx                ;
.text:00401CD7                 call    ebp ; send
.text:00401CD9                 cmp     eax, 0FFFFFFFFh
.text:00401CDC                 jz      loc_401C10
.text:00401CE2                 push    3E8h               ;
.text:00401CE7                 call    esi ; Sleep
00001082 0000000000401C82: ConnectC2+F2
```

```
; char SSL_Hello
SSL_Hello       db 16h                          ; DATA XREF: ConnectC2+ED↑o
                db    3
                db    1
                db    0
                db  6Bh ; k
                db    1
                db    0
                db    0
                db  67h ; g
                db    3
                db    1
                db  54h ; T
                db 0B4h ; ¦
                db 0C9h ; +
                db  7Bh ; {
                db  4Fh ; O
                db 0CFh ; ¤
                db 0BCh ; +
                db  5Ah ; Z
                db    1
                db 0ECh ; ý
                db  4Ah ; J
                db  73h ; s
                db 0C8h ; +
                db  6Dh ; m
                db 0BBh ; +
                db 0C0h ; +
                db  86h ; å
                db  9Fh ; █
                db  7Bh ; {
                db 0A9h ; ®
                db    8
                db  6Ah ; j
                db  60h ; `
                db  37h ; 7
```

In this situation it would appear that the author has simply copy & pasted values from a packet capture and in doing so they have revealed what is most likely the exact time and date that the packet was generated.



```
⊟ Random
    GMT Unix Time: Jan 13, 2015 15:30:03.000000000 China Standard Time
    Random Bytes: 4fcfbc5a01ec4a73c86dbbc0869f7ba9086a60370581971a...
  Session ID Length: 0
  Cipher Suites Length: 24
```

Interestingly there is one identifiable string in the data: www.apple.com. I think we have just discovered why this malware is called Mocelpa: ApleCom <> Mocelpa. This suggests that the individual who named the malware knew that the connection data was hardcoded and not actually encrypted. Those of you who are familiar with IDS/IPS and network detection will know that this behaviour makes a highly reliable signature. In all Mocelpa samples we analysed (see appendix for MD5's) this string remained the same.

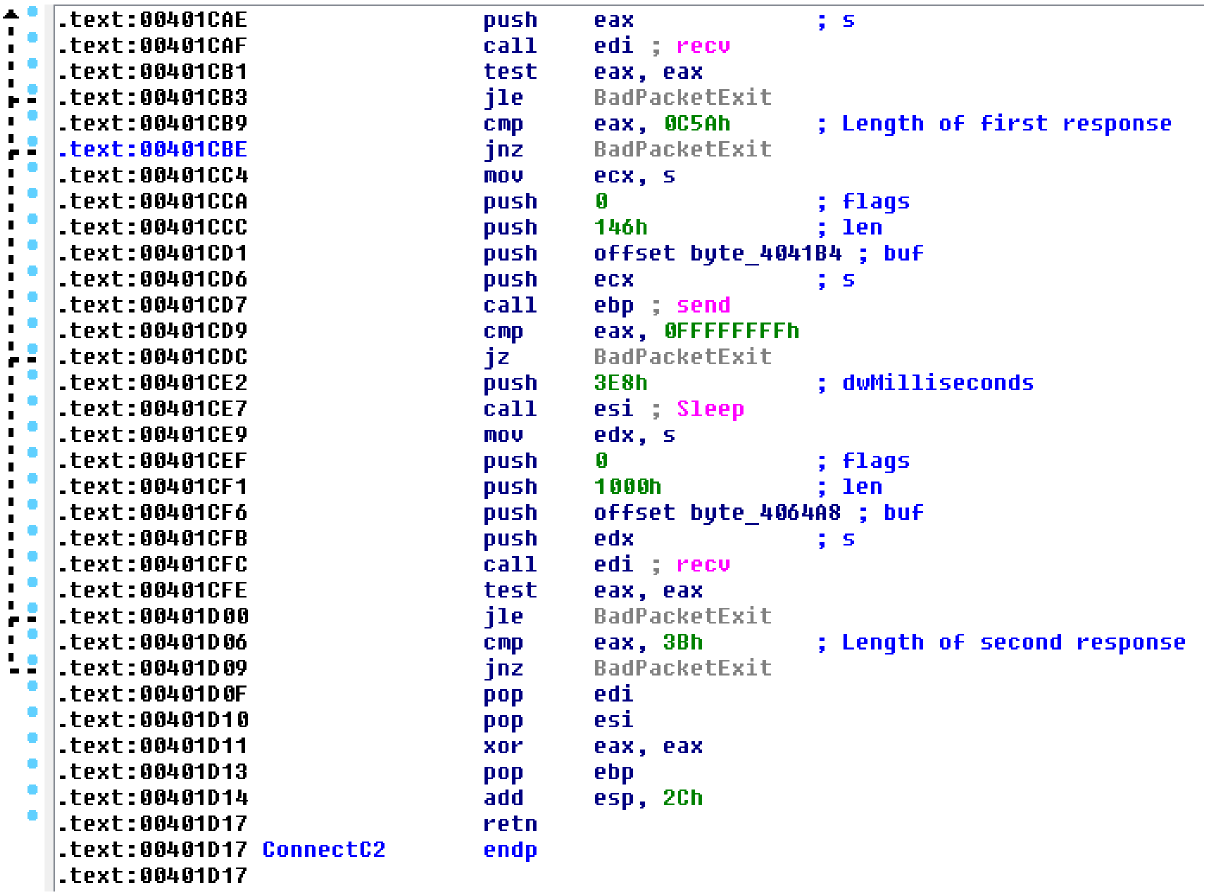
Now that we know the traffic data is hardcoded let's take a look at what follows the 'SSL' handsake.

Firstly, Mocelpa grabs the MAC address of the machine and runs it through an encoding routine. The encoding performed simply increases each number/character in the MAC address by 1. This value is then modified, by inserting hardcoded hexadecimal values at the beginning and end of the string.

Before connecting to the command and control server Mocelpa looks up the proxy server that is configured in Internet Explorer. This can be found in the registry under *"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"* in the *"ProxyServer"* key. It appears that at least one of the samples we analysed (see appendix) has a bug[3] and will fail to connect to the command and control server unless the system is configured to use a proxy server. Failure to connect to the command and control server results in the malware sleeping for 5 minutes before trying again.

[3] Thanks to Tillmann Werner for pointing this out

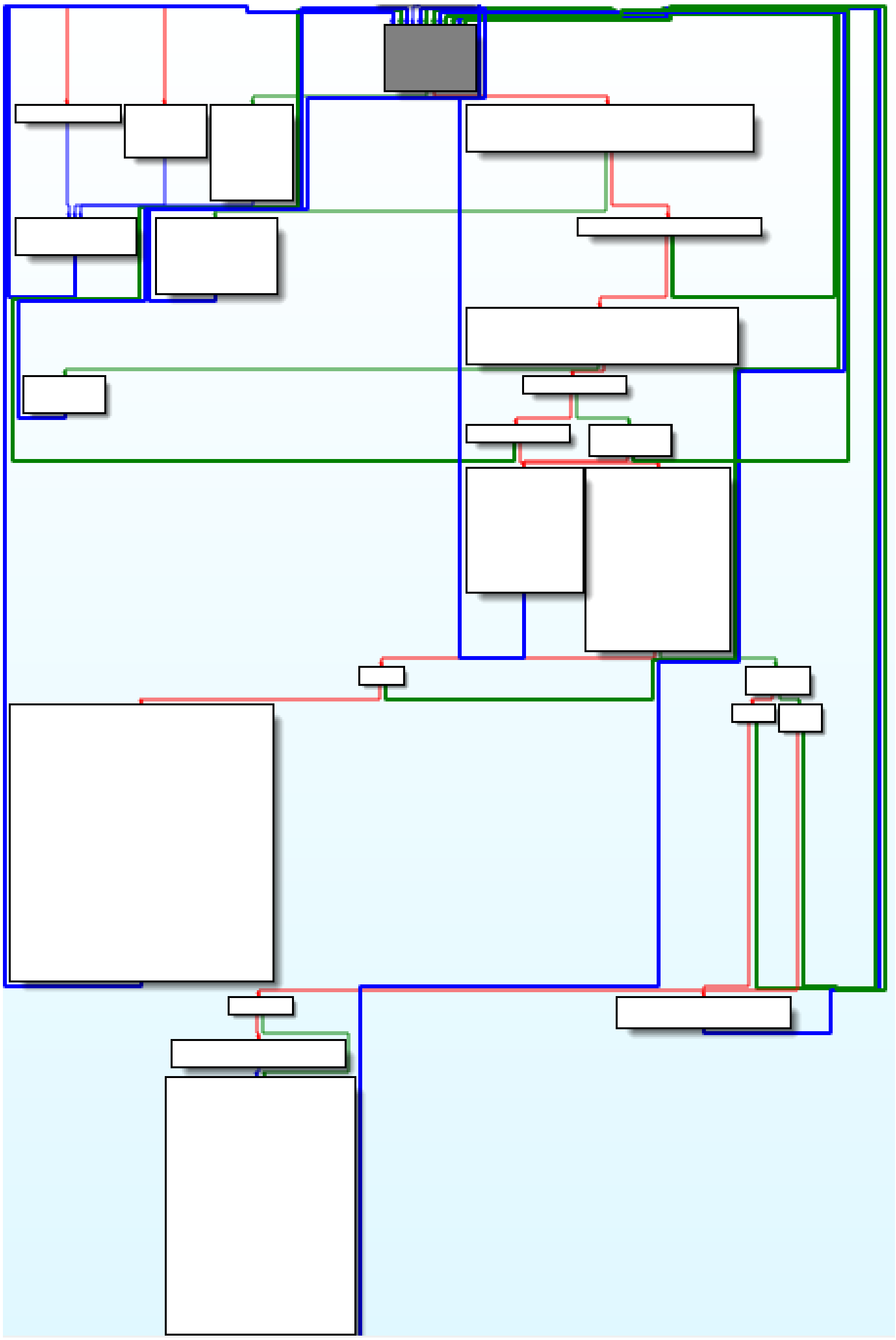
Upon connecting to the command and control server several exchanges of information take place. During the initial ‘SSL’ connection (described above) certain responses are expected from the command and control server. These responses are only validated by length and in fact can contain any data; the first response should be 3162 bytes in length and the second 59 bytes.



```
.text:00401CAE                  push    eax             ; s
.text:00401CAF                  call    edi ; recv
.text:00401CB1                  test    eax, eax
.text:00401CB3                  jle     BadPacketExit
.text:00401CB9                  cmp     eax, 0C5Ah      ; Length of first response
.text:00401CBE                  jnz     BadPacketExit
.text:00401CC4                  mov     ecx, s
.text:00401CCA                  push    0               ; flags
.text:00401CCC                  push    146h            ; len
.text:00401CD1                  push    offset byte_4041B4 ; buf
.text:00401CD6                  push    ecx             ; s
.text:00401CD7                  call    ebp ; send
.text:00401CD9                  cmp     eax, 0FFFFFFFFh
.text:00401CDC                  jz      BadPacketExit
.text:00401CE2                  push    3E8h            ; dwMilliseconds
.text:00401CE7                  call    esi ; Sleep
.text:00401CE9                  mov     edx, s
.text:00401CEF                  push    0               ; flags
.text:00401CF1                  push    1000h           ; len
.text:00401CF6                  push    offset byte_4064A8 ; buf
.text:00401CFB                  push    edx             ; s
.text:00401CFC                  call    edi ; recv
.text:00401CFE                  test    eax, eax
.text:00401D00                  jle     BadPacketExit
.text:00401D06                  cmp     eax, 3Bh        ; Length of second response
.text:00401D09                  jnz     BadPacketExit
.text:00401D0F                  pop     edi
.text:00401D10                  pop     esi
.text:00401D11                  xor     eax, eax
.text:00401D13                  pop     ebp
.text:00401D14                  add     esp, 2Ch
.text:00401D17                  retn
.text:00401D17 ConnectC2        endp
.text:00401D17
```

The encoded MAC address is also sent to the command and control server. It is likely that the attacker’s use this value to identify unique victims. Once the connection has been established command and control can take place.

The command and control functionality is very simple, however the structure is not. From this very high-level view we can see that this is not as easy as a simple IF, NOT, THEN structure.

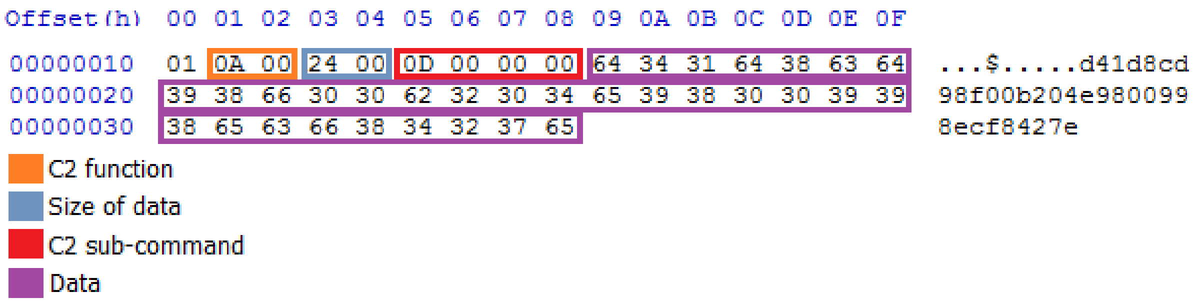


Ultimately this breaks to commands and sub-commands, denoted by a specific packet structure.

The functionality can be broken down into the following groups:

- **0x3: Execute command**
  - Uses cmd.exe to execute a command, logs the output and sends it back to the c2 server
- **0x0A: File operations**
  - Sub-options:
    - **0x0B:** Create a file (filename specified by in the data packet) in the user's temporary directory
    - **0x0C:** Write data (specified in the data packet) to the currently open file
    - **0x0D:** Open the file that was created/written to, perform an MD5 hash on the contents and compare it to the attacker specified MD5 (contained within the data packet)

The structure behind the protocol is fairly simple and involves packet data being passed through a decoding routine before being processed. For example, a decoded packet executing C2 command 0x0A, sub-command 0x0D looks like the following:

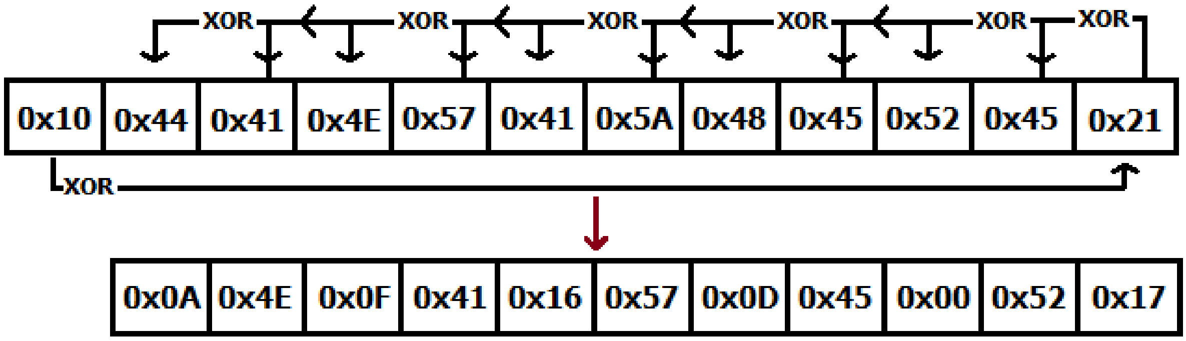


```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000010   01 0A 00 24 00 0D 00 00 00 64 34 31 64 38 63 64   ...$.....d41d8cd
00000020   39 38 66 30 30 62 32 30 34 65 39 38 30 30 39 39   98f00b204e980099
00000030   38 65 63 66 38 34 32 37 65                        8ecf8427e
```

🟧 C2 function
🟦 Size of data
🟥 C2 sub-command
🟪 Data

The data decoding routine begins by taking a series hardcoded values and appending it to the beginning of the data, it then proceeds to run an XOR operation against each byte of the packet in reverse order, for example:



0x10 | 0x44 | 0x41 | 0x4E | 0x57 | 0x41 | 0x5A | 0x48 | 0x45 | 0x52 | 0x45 | 0x21

0x0A | 0x4E | 0x0F | 0x41 | 0x16 | 0x57 | 0x0D | 0x45 | 0x00 | 0x52 | 0x17

Clearly at this stage we can see that this implant does not use SSL.
Server-side protocol analysis has unfortunately been thwarted by a lack of response from the command and control servers so the examples are above are just that: examples. In any case we are

confident in saying that responses from live command and control servers are unlikely to be fully static and thus creating a reliable IDS signature or detection heuristic would be challenging at best.

During the process of analysing the malware samples we created a fully functioning command and control server module. After careful consideration we have decided not to release the code for this.

Ultimately Mocelpa is a simple downloader with basic functionality. Given the seemingly unnecessary amount of complexity involved and the obscure method of verifying file download success we would be quick to assume that this was written by an inexperienced programmer, but there are in fact a number of reasons why such methods were used in the development of this implant. Taking into consideration things like reverse engineering, network detection devices, anti-virus and human ‘hunter’ teams it does not take much thought to theorise why the codebase and functionality are somewhat creative.

Still, through this extra code it does not make Mocelpa less detectable – in fact quite the opposite.

Detection & mitigation

This implant can be detected at both disk and network level. In order to help organisations protect themselves we have created a number of network IDS rules and disk-scan rules that can be used with Snort and Yara. Rules are provided in a best-effort basis and we cannot vouch for their efficiency in your environment.

Mocelpa YARA disk signature

```
rule apt_win_mocelpa {
meta:
      author = "@int0x00"
      description = "APT malware; Mocelpa, downloader."
strings:
      $mz = {4D 5A}
      $ssl_hello = {16 03 01 00 6B 01 00 00 67 03 01 54 B4 C9 7B 4F
CF BC 5A 01 EC 4A 73 C8 6D BB C0 86 9F 7B A9 08 6A 60 37 05 81 97 1A
C8 9F 45 E5 00 00 18 00 2F 00 35 00 05 00 0A C0 13 C0 14 C0 09 C0 0A
00 32 00 38 00 13 00 04 01 00 00 26 00 00 00 12 00 10 00 00 0D 77 77
77 2E 61 70 70 6C 65 2E 63 6F 6D 00 0A 00 06 00 04 00 17 00 18 00 0B
00 02 01 00}
condition:
      ($mz at 0) and ($ssl_hello)
}
```

Mocelpa SNORT network beaconing

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 443 (msg:"APT MALWARE –
Mocelpa beacon"; flow:established,to_server; content:"|16 03 01 00
6B 01 00 00 67 03 01 54 B4 C9 7B 4F CF BC 5A 01 EC 4A 73 C8 6D BB C0
86 9F 7B A9 08 6A 60 37 05 81 97 1A C8 9F 45 E5 00 00 18 00 2F 00 35
00 05 00 0A C0 13 C0 14 C0 09 C0 0A 00 32 00 38 00 13 00 04 01 00 00
26 00 00 00 12 00 10 00 00 0D 77 77 77 2E 61 70 70 6C 65 2E 63 6F 6D
00 0A 00 06 00 04 00 17 00 18 00 0B 00 02 01 00|"; classtype:trojan-
activty; sid:YOUR_SID; rev:14062015)
```

Mocelpa SNORT C2 server IP #1

```
alert ip $HOME_NET any <> 213.179.57.178 any (msg:"APT MALWARE –
Mocelpa C2 address"; classtype:trojan-activity; sid:YOUR_SID; rev:
14062015;)
```

Mocelpa SNORT C2 server IP #2

```
alert ip $HOME_NET any <> 128.91.34.188 any (msg:" APT MALWARE –
Mocelpa C2 address"; classtype:trojan-activity; sid:YOUR_SID; rev:
14062015;)
```

Mocelpa SNORT C2 server IP #3

```
alert ip $HOME_NET any <> 200.87.48.4 any (msg:"APT MALWARE –
Mocelpa C2 address"; classtype:trojan-activity; sid:YOUR_SID; rev:
14062015;)
```

Mocelpa SNORT C2 server IP #4

```
alert ip $HOME_NET any <> 128.91.34.175 any (msg:"APT MALWARE –
Mocelpa C2 address"; classtype:trojan-activity; sid:YOUR_SID; rev:
14062015;)
```

Appendix

The following artefacts were found during the investigation

MD5s	Network artefacts
6e4e030fbd2ee786e1b6b758d5897316	213.179.57.178
27f5b6e326e512a7b47e1cd41493ee55[4]	128.91.34.188
548884eabebef0081dd3af9f81159754	128.91.34.175
05bc4a9b603c1aa319d799c8fba7a42a	200.87.48.4
cdf0e90b0a859ef94be367fdd1dd98c6	

[4] “Broken”; will not connect to the internet unless a system proxy is configured

Contact

For all questions relating to the publication or specifics in this document please contact us via one of the following methods:

Twitter: @dragonthreatlab
Website: http://dragonthreat.blogspot.hk
Email: dragonthreatlabs@gmail.com

Kind regards,

Dan (@int0x00)
Dragon Threat Labs