Black Energy Crypto

web.archive.org/web/20140428201836/http://www.fireeye.com/blog/technical/malware-research/2010/03/black-energy-crypto.html

By Julia Wolf

March 3, 2010

Introduction

Black Energy has been in the news again recently (well, *it was recent* back when I wrote the first draft of this).

I'm not here to talk about Citigroup, I'm here to talk about cryptography, and how to fail at it. That being said, allegedly Citibank was "hacked" using Black Energy, according to the Wall Street Journal. Citigroup flat out denies it, and aside from this assertion from the WSJ, there's no other information. But it doesn't make sense that "Black Energy" itself, or what is commonly referred to by that name, was used for some kind of banking attack; It's a DDoS bot.

Now, it could actually be Black Energy that's responsible, or something different which just looks like Black Energy. But lately, a very Black Energy-like DDoS "module" tends to get installed along with other malware such as Zeus, via the "Yes Exploit System", or via Oficla/Sasfis, and like every bot, it can download and execute arbitrary files upon command. I have no idea what, if anything, happened at Citibank, but I speculate that a Black Energy bot was just along for the ride. An infection of one bot, quickly leads to an infestation of many. [cute metaphor about infestations goes here] It's kinda like a big ball of malware goo.

Analogy

Ok, so you remember how the five robot lions in the show "Voltron" would form a giant robot to battle space monsters? Each lion had its own distinct identity, like one was green, and another one was pink, etc. but they could combine to form a single robot, with a distinct identity apart from each individual lion. Ok, well malware also combines together to form a giant robot.

[I was going to make the same analogy using the Constructicons as examples, they're evil bots you see… but that's just a little too obscure.]

Anyway, so for something less ambiguous… onto the technical part!

The Technical Part

Exposition

I'm never certain about how much information I should publicly reveal about how much is known about a particular malware. It's very ego gratifying to say: Hey, the way you did this thing sucks.

I'm certain that word will get back to the original malware author(s), and they'll fix the bug(s).

So, in the case of Black Energy, or the bot formerly known as Black Energy, someone [Cr4sh?] already got a clue that they were doing it wrong, and fixed it. So this crypto trick I'm about to reveal below doesn't work anymore.

The 1.x versions of Black Energy just encoded their communications with Base64.

Jose Nazario wrote a good paper on it here:
BlackEnergy DDoS Bot Analysis.pdf [PDF]

And some other stuff about it: Posts tagged Black Energy

And I'm sure that Joe Stewart has probably said something about it somewhere, sometime.

Update

Cr4sh — author of the original Black Energy 1.x bot(s) — released a public statement, for those who can't read Russian (or read it less well than even I do), I'll summarize the relivant parts:

- Black Energy was created as a simple DDoS bot without any rootkits, infectors, spyware, etc.
- Cr4sh doesn’t how how this relates to stealing banking information.
- The source code was available to many private parties, so someone must have modified it.
- Only an idiot would put their name on a criminal bot.
- Cr4sh says he’s not linked to any botnet networks, programs, or organizations.

> Black Energy – это DDoS-бот, который был результатом проведенной на заказ (году в 2006-2007) работы, по созданию простого и компактного рабочего инструмента, без всяких руткитов, инфекторов и каких бы то ни было spyware-функций.

> Каким образом DDoS-бот относится к краже банковской информации – я в душе не ебу. Однако, тот факт, что его исходники были доступны многим людям во всевозможных [полу]приватных тусовках, может означать, что кто-то заточил его под свои нужды. Подозревать же в причастности к криминальным махинациям автора бота, чей автограф стоял на публично доступных билдах 3-х летней давности, может только полный идиот.

> PS: я же в настоящее время никаким боком не связан с ботнет-сетями и программами для их организации.

Exposition

Someone sent me a .pcap of some recent encrypted Black Energy communications; And had an urgent need to know what was being transmitted. But they didn’t have a copy of the bot which spewed forth this data, and the associated C&C was down.

Well, maybe it was this sample:

45f3085144f6875b27ae173f22856198 [Live Sample] [Anubus] [Another Anubus] [ThreatExpert]

Or this one:

0eb19a4f7f3de5a2bb6460230f7ddfe4 [Live Sample] [Anubus]
[Another Anubus] [ThreatExpert]

Or this one:

[etc.]
In any case, sometimes it’s usually a bit faster to just attack the crypto head-on, rather than reverse engineering a whole program looking for the decryption routine. This is because most malware authors (and sadly, many commercial software authors too) implement the crypto routine(s) incorrectly, making them fairly trivial to crack. So, rather than track down a sample and reversing it, I just did some math on the cyphertexts, looking for common mistakes.

The Actual Technical Part

There was nothing known [to me or Google] about the encryption being used. But I noticed some very strong patterns in the data. It was obviously some kind of stream cipher, but not RC4 from what I can tell. I ran all the usual tests (`add`, `subtract`, or `xor` with repeating or incrementing patterns), it wasn’t one of those.

Stream Cypher

This version of Black energy is reusing the same keystream for multiple plaintexts. This is an absolutely fatal flaw for any stream cypher; Take a look at this example to see why. I say it’s obviously a stream cypher with a constant key, because if you `XOR` two cyphertexts together, the keysteam cancels out, and at every location there was a byte in one plaintext, which is the same as the byte at the same location in the other, you get a `NULL`. And transpositionally, if there was a `NULL` byte in either plaintext, the plaintext byte from the opposite plaintext will emerge.

Remember this identity: $(Message_1 \oplus Keystream) \oplus (Message_2 \oplus Keystream) = Message_1 \oplus Message_2$

For Example

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000030  00 00 00 00 00 00 00 00  00 00 00 00 10 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  8a d1 0b 61 8a d1 0b 61  8a d1 0b 61 8a d1 0b 61  |...a...a...a...a|
00000090  8a d1 59 61 8b d1 0b 61  4c d2 09 61 89 d1 0b 61  |..Ya...aL..a...a|
000000a0  9c d5 0b 61 8a d1 0b 61  8a d1 09 61 80 d1 0b 61  |...a...a...a...a|
000000b0  09 d9 57 61 ab d1 0b 61  8f dd 6a 61 9e d1 0b 61  |..Wa...a..ja...a|
000000c0  8a d1 37 61 8e d1 0b 61  8a d1 0d 61 8a d1 0b 61  |..7a...a...a...a|
000000d0  fe ba 90 af 8b d1 0b 61  52 69 63 68 a9 df a9 c7  |.......aRich....|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
000000f0  79 42 b3 48 00 00 00 00  50 45 00 00 ac 01 0a 21  |yB.H....PE.....!|
00000100  9e 27 a1 42 00 0e 00 00  00 3a 00 00 e0 00 0e 21  |.'.B.....:.....!|
00000110  a7 11 07 0a 00 1e 00 00  00 30 00 00 00 00 00 10  |.........0......|
00000120  41 00 00 00 00 12 00 00  04 20 00 00 00 00 00 10  |A........ ......|
00000130  04 10 00 00 00 02 00 00  04 70 00 00 00 04 00 00  |.........p......|
00000140  04 00 00 00 02 00 00 04  00 50 10 00 00 14 00 00  |.........P......|
00000150  00 00 10 00 02 10 00 04  00 00 10 00 10 10 00 00  |................|
00000160  10 26 10 00 62 10 00 00  08 22 00 00 a4 00 00 00  |.&..b...."......|
00000170  10 23 00 00 63 00 00 00  00 21 00 00 64 00 00 00  |.#..c....!..d...|
00000180  00 00 00 00 00 00 00 00  00 60 00 00 bc 00 00 00  |.........`......|
00000190  00 00 00 00 00 00 00 00  00 40 00 00 20 01 00 00  |.........@.. ...|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001c0  00 20 00 00 b4 00 00 00  00 00 00 00 00 00 00 00  |. ..............|
000001d0  00 20 00 00 5c 00 00 00  00 00 00 00 00 00 00 00  |. ..\...........|
000001e0  2e 74 65 78 74 00 00 00  00 0d 00 00 00 10 00 00  |.text...........|
000001f0  2e 7a 65 78 74 04 00 00  dd 0d 00 00 00 10 00 00  |.zext...........|
00000200  00 0e 00 00 20 04 00 60  2e 72 64 61 74 61 00 00  |.... ..`.rdata..|
00000210  72 06 00 00 20 20 00 60  2e 7a 64 61 74 73 00 00  |r...  .`.zdats..|
00000220  73 03 00 00 00 20 00 00  00 04 00 00 40 12 00 40  |s.... ......@..@|
00000230  2e 64 61 74 61 00 00 00  70 2e 00 00 40 30 00 40  |.data...p...@0.@|
00000240  2e 42 61 74 61 1a 00 00  f0 09 00 00 00 30 00 00  |.Bata........0..|
00000250  00 02 00 00 40 16 00 c0  2e 72 65 6c 6f 63 00 00  |....@....reloc..|
00000260  4c 01 00 00 40 60 00 c0  2e 70 65 6c 6f 23 00 00  |L...@`...pelo#..|
00000270  86 01 00 00 00 40 00 00  00 02 00 00 40 18 00 42  |.....@......@..B|
00000280  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 42  |............@..B|
00000290  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Wow, that sure does look like two EXE’s XOR’d together, doesn’t it?

Step By Step Example

Since this Anubus report is public, I’ll use it for my example:

http://anubis.iseclab.org/?action=result&task_id=15921f68dbf4f3be4255a4705efd2451f

Grab the .pcap file

http://anubis.iseclab.org/?action=result&task_id=15921f68dbf4f3be4255a4705efd2451f&download=traffic.pcap

And extract all of the TCP streams from this .pcap into individual files, with your favorite program for doing that kind of thing. (I think Wireshark will do it; I just always use my own Perl scripts cause it’s fast and automated.) Stripped of HTTP headers, theses are the files I got:

MD5	Timestamp	Source IP:Port	Destination IP:Port	Size	Description
3e736a6d3f8fe6cf7e54a7658cba9352	1253064020.308867	88.214.243.45:80	192.168.0.2:1038	785	Encrypted XML
5394487c93a748e6b0b182101ba56a56	1253064023.563605	88.214.243.45:80	192.168.0.2:1039	11264	Encrypted “ddos”
d15af19966a4782ede44a1e62f8cf70b	1253064029.663532	88.214.243.45:80	192.168.0.2:1040	6657	Encrypted “http”
5730dbddc77de80d3d7e053699cb2136	1253064034.512609	88.214.243.45:80	192.168.0.2:1041	16896	Encrypted “syn”

So, since EXE files are mostly NULLs at the beginning… And the “syn” one seems to have the most NULLs at the beginning… It’s the one that yields the most amount of printable text.

```
hexdump -C Config_x_syn
00000000  71 65 e8 6d 6f 20 76 65  76 73 69 6f 91 c2 22 31  |qe.mo vevsio.."1|
00000010  96 30 22 3f 3e 0a 3c 62  2b 65 72 6e 65 6c 3e 0a  |.0"?>.<b+ernel>.|
00000020  3c 70 6c 75 67 69 6e 73  3e 0a 3c 70 6c 75 67 69  |<plugins>.<plugi|
00000030  6e 3e 0a 3c 6e 61 6d 65  3e 64 64 6f 9b 3c 2f 6e  |n>.<name>ddo.</n|
00000040  6f 72 df 30 0a 88 7f a8  53 cb 68 23 a3 1f 65 54  |or.0....S.h#..eT|
00000050  46 05 45 02 01 06 08 1c  5f 67 1c 4c 11 02 1b 08  |F.E....._g.L....|
00000060  1d 4e 5c 6f 1c 02 19 1b  47 00 00 1e 4e 73 3d 41  |.N\o....G...Ns=A|
00000070  00 0a 5a 0d 5a 79 7d 36  0b 6e 61 6d 65 3e 0a 3c  |..Z.Zy}6.name>.<|
00000080  11 0a be 86 4a 61 cc 98  12 32 8d d0 46 7c d1 cf  |....Ja...2..F|..|
00000090  49 6c c1 ac 1d 21 d2 ca  95 66 96 c8 1f 04 9e d6  |Il...!...f......|
000000a0  5c 73 98 cf 4f 30 a8 9a  ce 67 92 c3 16 7d db c8  |\s..O0...g...}..|
000000b0  1f 21 cd c7 68 6b 9c ac  1a 74 a7 d4 57 67 cd c8  |.!..hk...t..Wg..|
000000c0  18 33 c2 89 54 6b d0 d5  4f 6d 96 98 28 32 8d d6  |.3..Tk..Om..(2..|
000000d0  3e 1c 04 01 4d 30 a8 9a  2f 70 6c 75 67 69 6e 73  |>...M0../plugins|
000000e0  3e 0a 3c 63 6d 64 73 3e  5a 79 2f 63 21 65 77 3e  |>.<cmds>Zy/c!ew>|
000000f0  73 7e c3 24 67 5f 64 61  74 61 3e 0a dc 64 6a 4e  |s~.$g_data>..djN|
00000100  78 3f 3b 7e 63 7e 5f 73  69 40 65 3e 31 30 30 30  |x?;~c~_si@e>1000|
00000110  90 3f 74 63 70 4f 73 69  7a 45 3e 0d 0a 3c 74 73  |.?tcpOsizE>..<ts|
00000120  70 4f 66 72 65 73 3e 35  34 3c 2f 74 63 70 5f 66  |pOfres>54</tcp_f|
00000130  76 65 71 3e 0d 0a 3c 74  63 00 5f 74 68 76 65 61  |veq>..<tc._thvea|
00000140  64 73 3e 35 3e 2f 74 67  70 5f 64 68 72 75 61 64  |ds>5>/tgp_dhruad|
00000150  73 3e 1d 0a 3c 65 64 70  5f 73 69 7a 75 3e 31 30  |s>..<edp_sizu>10|
00000160  20 16 3c 2f 17 64 70 5f  7b 4b 7a 65 8a 0d 0a 3c  | .</.dp_{Kze...<|
00000170  75 64 70 5f 66 72 65 71  3e 35 30 3c 2f 75 64 70  |udp_freq>50</udp|
00000180  5f 66 72 65 71 3e 0d 0a  3c 15 64 70 e3 74 68 72  |_freq>..<.dp.thr|
00000190  65 61 64 73 3e 35 3c 2f  75 64 70 5f 74 68 72 65  |eads>5</udp_thre|
000001a0  61 64 73 3e 0d 0a 3c 69  63 6d 70 5f 73 69 7a 65  |ads>..<icmp_size|
000001b0  3e 31 30 30 30 3c 2f 69  63 6d 70 5f 73 69 7a 65  |>1000</icmp_size|
000001c0  3e 2d 0a 3c dd 63 6d 70  5f 66 72 65 71 3e 35 30  |>-.<.cmp_freq>50|
000001d0  3c 2f 69 63 6d 70 5f 66  72 65 71 3e 0d 0a 3c 69  |</icmp_freq>..<i|
000001e0  4d 19 15 27 00 68 72 65  61 69 73 3e 35 2c 2f 69  |M..'.hreais>5,/i|
000001f0  63 63 70 5f 74 6c 72 65  61 64 73 3e 0d 0a 3c 68  |ccp_tlreads>..<h|
00000200  74 74 70 5f 46 72 65 11  10 43 54 51 48 4e 68 74  |ttp_Fre..CTQHNht|
00000210  06 76 5f 66 72 45 71 3e  0d 02 3c 68 74 66 70 5f  |.v_frEq>..<htfp_|
00000220  74 68 72 65 61 64 73 3e  35 3c 2f 68 34 74 70 1f  |threads>5</h4tp.|
00000230  5a 0c 13 11 00 64 73 3e  4c 01 64 64 6f 43 3e 0a  |Z....ds>L.ddoC>.|
00000240  3c 4e 74 74 70 24 3c 68  74 74 70 5f 66 72 65 71  |<Nttp$<http_freq|
00000250  3e 33 30 3c 6f 68 74 b4  5e 2d 03 1e 0a 12 3e 0d  |>30<oht.^-....>.|
00000260  46 3d 68 74 74 10 5f 74  68 70 65 61 64 33 3e 32  |F=htt._thpead3>2|
00000270  3c 2f 68 74 74 70 5f 74  68 72 65 61 24 73 3e 7e  |</http_threa$s>~|
00000280  2f 68 74 74 70 3e 0a 3c  73 79 6e 3e 3c 73 79 6e  |/http>.<syn><syn|
00000290  5f 66 72 65 71 3e 32 30  3c 2f 73 79 6e 5f 66 72  |_freq>20</syn_fr|
000002a0  65 71 3e 0d 0a 3c 73 79  6e 5f 74 68 72 65 61 64  |eq>..<syn_thread|
000002b0  73 3e 33 3c 2f 73 79 6e  5f 74 68 72 65 61 64 73  |s>3</syn_threads|
000002c0  3e 3c 2f 73 79 6e 3e 0a  3c 2f 70 6c 67 5f 64 61  |></syn>.</plg_da|
000002d0  74 61 3e 0a 3c 73 6c 65  65 70 66 72 65 71 3e 39  |ta>.<sleepfreq>9|
000002e0  30 30 3c 2f 73 6c 65 65  70 66 72 65 71 3e 0a 3c  |00</sleepfreq>.<|
000002f0  69 70 3e 31 32 38 2e 31  33 30 2e 35 36 2e 32 32  |ip>128.130.56.22|
00000300  3c 2f 69 70 3e 0a 3c 2f  62 6b 65 72 6e 65 6c 3e  |</ip>.</bkernel>|
00000310  0a                                                |.|
00000311
md5sum Config_x_syn
0880df84c5b886cbdd0e0be01deed2f6  Config_x_syn
```

Well, that certainly looks like XML doesn’t it?

Fixup the XML

You can also approach this from the EXE side of things, but most of you are probably better at reading XML than EXE. XML also provides for a lot of redundant plaintext.

General Procedue

Since it’s XML, you know every tag must match up, so one can fix the broken tags easily.

- Merge the all of the known plaintext you can find from the config files together and make the obvious fixes.
- `XOR` this back against the original cyphertext, and you get a big chunk of the keystream.
- Then `XOR` this partial keystream against the EXEs.
- Fix up the EXE headers, so that most of the fields make sense, try to use the RICH header if present, and make sure “.text”, “.data”, “.reloc” are spelled correctly, etc.

- If you have a large malware zoo, you can look for EXE files from September, that are 11264, 6657, or 16896 bytes long, and match up mostly within the first few hundred bytes to this.

  (The quick way is to just XOR the plaintext you've got so far, against the other plaintext samples in your zoo, and see which one(s) have the lowest entropy.
  It's probably one of those. Otherwise you can just try them all if it's not too many (and in this case it's not too many).)
- If you don't have a sample of one of the EXEs, then just re- `XOR` again the EXE plaintext you've recovered so far with the original cyphertext.
- And then `XOR` that partial keystream back against the config file.
- Do the same sort of fix-ups you did before (you've got some more bytes now, the ones you just fixed in the EXE).
- Then re- `XOR` this back the other way, lather rinse repeat, until you're down to just the bits you don't know between all four of the plaintexts.

At this point, you'll have recovered most of the plaintext/keystream, without knowing the password *or even the cypher used*! This trick works for any cypher that is basically `$randomness ^ $plaintext` , where `$randomness` gets used more than once. There's a reason the one-time-pad is *one-time*-only.

Step-by-step Example

If you don't have a hex editor, you can use any 8-bit clean text editor to make the changes, be careful not to add or remove any characters, otherwise everything after that point will be shifted. (I use Joe as my editor, believe it or not.)

[By the way, if you think this part looks like it took me a long time to write. This was actually the least time-intensive part. The introductory paragraphs up at the beginning took the longest to write.]

```
00000000  71 65 e8 6d 6f 20 76 65  76 73 69 6f 91 c2 22 31  |qe.mo vevsio.."1|
00000010  96 30 22 3f 3e 0a 3c 62  2b 65 72 6e 65 6c 3e 0a  |.0"?>.<b+ernel>.|
…
00000300  3c 2f 69 70 3e 0a 3c 2f  62 6b 65 72 6e 65 6c 3e  |</ip>.</bkernel>|
```

- We know the first two bytes of the EXE are " `MZ` ", so `XOR` " `qe` " with " `MZ` " to get " `<?` "
- But really, that whole first line can't be anything other than " `<?xml version="1.0"?>` "
- There is a " `</bkernel>` " tag at the end, so that must be a " `<bkernel>` " tag at the beginning.

```
00000030  6e 3e 0a 3c 6e 61 6d 65  3e 64 64 6f 9b 3c 2f 6e  |n>.<name>ddo.</n|
00000040  6f 72 df 30 0a 88 7f a8  53 cb 68 23 a3 1f 65 54  |or.0....S.h#..eT|
…
00000070  00 0a 5a 0d 5a 79 7d 36  0b 6e 61 6d 65 3e 0a 3c  |..Z.Zy}6.name>.<|
…
000000d0  3e 1c 04 01 4d 30 a8 9a  2f 70 6c 75 67 69 6e 73  |>...M0../plugins|
```

- It's safe to assume that " `ddo\x9B` " is really " `ddos` ".
- The closing tag " `</nor\xDF0` " must be " `</name>` "
- And further down, " `6\x0Bname>` " can be either " `<name>` " or " `</name>` ". We'll figure it out later.
- " `\x9A/plugins>` " is `"</plugins>` ".

```
000000f0  73 7e c3 24 67 5f 64 61  74 61 3e 0a dc 64 6a 4e  |s~.$g_data>..djN|
00000100  78 3f 3b 7e 63 7e 5f 73  69 40 65 3e 31 30 30 30  |x?;~c~_si@e>1000|
00000110  90 3f 74 63 70 4f 73 69  7a 45 3e 0d 0a 3c 74 73  |.?tcpOsizE>..<ts|
00000120  70 4f 66 72 65 73 3e 35  34 3c 2f 74 63 70 5f 66  |pOfres>54</tcp_f|
00000130  76 65 71 3e 0d 0a 3c 74  63 00 5f 74 68 76 65 61  |veq>..<tc._thvea|
00000140  64 73 3e 35 3e 2f 74 67  70 5f 64 68 72 75 61 64  |ds>5>/tgp_dhruad|
00000150  73 3e 1d 0a 3c 65 64 70  5f 73 69 7a 75 3e 31 30  |s>..<edp_sizu>10|
```

- " `djNx?;~c~_si@e>1000\x90?tcpOsizE>` " looks like a pair of tags named " `tcp_size` ", so that becomes:

  " `djNx?<tcp_size>1000</tcp_size>` "

- Between " `<tspOfres>54</tcp_fveq>` " looks like a pair of tags named " `tcp_freq` ", so this becomes:

  " `<tcp_freq>54</tcp_freq>` "
- " `<tic\x00_thveads>5>/tgp_dhruads>` " looks like it says " `tcp_threads` ", so this becomes:
  " `<tcp_threads>5</tcp_threads>` ", and

- the end of line should be “ `\x0D\x0A` ” rather than “ `\x1D\x0A` “

```
00000150  73 3e 1d 0a 3c 65 64 70  5f 73 69 7a 75 3e 31 30  |s>..<edp_sizu>10|
00000160  20 16 3c 2f 17 64 70 5f  7b 4b 7a 65 8a 0d 0a 3c  | .</.dp_{Kze...<|
00000170  75 64 70 5f 66 72 65 71  3e 35 30 3c 2f 75 64 70  |udp_freq>50</udp|
00000180  5f 66 72 65 71 3e 0d 0a  3c 15 64 70 e3 74 68 72  |_freq>..<.dp.thr|
```

- “ `<edp_sizu>10 \x16</\x17dp_{Kze\x8A` ” looks like it says “ `udp_size` ” (This follows a naming pattern established by “ `tcp_size` ” above)

  So, fixing those tags:
  “ `<udp_size>10 \x16</udp_size>` “, and if it’s following the same pattern as “ `tcp_size` ” above, then the number within the tags is probably “ `1000` “
  “ `<udp_size>1000</udp_size>` “
- “ `<udp_freq>50</udp_freq>` ” That looks just right… it also implies that the “ `<tcp_freq>54</tcp_freq>` ” from above should really be:
  “ `<tcp_freq>50</tcp_freq>` “

```
00000180  5f 66 72 65 71 3e 0d 0a  3c 15 64 70 e3 74 68 72  |_freq>..<.dp.thr|
00000190  65 61 64 73 3e 35 3c 2f  75 64 70 5f 74 68 72 65  |eads>5</udp_thre|
000001a0  61 64 73 3e 0d 0a 3c 69  63 6d 70 5f 73 69 7a 65  |ads>..<icmp_size|
000001b0  3e 31 30 30 30 3c 2f 69  63 6d 70 5f 73 69 7a 65  |>1000</icmp_size|
```

- “ `<\x15dp\xE3threads>5</udp_threads>` ” should obviously be:

  “ `<udp_threads>5</udp_threads>` “, and the number matches with “ `tcp_threads` ” above.
- “ `<icmp_size>1000</icmp_size>-` ” just needs the “ `-` ” changed back to a “ `\x0D` ” end of line character.

```
000001c0  3e 2d 0a 3c dd 63 6d 70  5f 66 72 65 71 3e 35 30  |>-.<.cmp_freq>50|
000001d0  3c 2f 69 63 6d 70 5f 66  72 65 71 3e 0d 0a 3c 69  |</icmp_freq>..<i|
000001e0  4d 19 15 27 00 68 72 65  61 69 73 3e 35 2c 2f 69  |M..'.hreais>5,/i|
000001f0  63 63 70 5f 74 6c 72 65  61 64 73 3e 0d 0a 3c 68  |ccp_tlreads>..<h|
00000200  74 74 70 5f 46 72 65 11  10 43 54 51 48 4e 68 74  |ttp_Fre..CTQHNht|
```

- “ `<\xDDcmp_freq>50</icmp_freq>` ” is obviously “ `<icmp_freq>50</icmp_freq>` “

- Based on everything so far, we can say that

  “ `<iM\x19\x15'\x00hreais>5,/iccp_tlreads>` ” is really:

  “ `<icmp_threads>5</icmp_threads>` “

```
00000200  74 74 70 5f 46 72 65 11  10 43 54 51 48 4e 68 74  |ttp_Fre..CTQHNht|
00000210  06 76 5f 66 72 45 71 3e  0d 02 3c 68 74 66 70 5f  |.v_frEq>..<htfp_|
```

  “ `<http_Fre\x11\x10CTQHNht\x06v_frEq>\x0D\x02` ” is clearly:

  “ `<http_freq>CTQ</http_freq>\x0D\x0A` ” well, not quite so clearly, it looks like there’s a three digit number that goes there, not 1000 as above

```
00000220  74 68 72 65 61 64 73 3e  35 3c 2f 68 34 74 70 1f  |threads>5</h4tp.|
00000230  5a 0c 13 11 00 64 73 3e  4c 01 64 64 6f 43 3e 0a  |Z....ds>L.ddoC>.|
```

- “ `<htfp_threads>5</h4tp\x1FZ\x0C\x13\x11\x00ds>Lx01ddoC>` ” Well, this first part is obvious:

  “ `<http_threads>5</http_threads>L\x01ddoC>` “
- And then that’s probably the closing tag to “ `<ddos>` “, so we get:
  “ `<http_threads>5</http_threads></ddos>` “

```
00000240  3c 4e 74 74 70 24 3c 68  74 74 70 5f 66 72 65 71  |<Nttp$<http_freq|
00000250  3e 33 30 3c 6f 68 74 b4  5e 2d 03 1e 0a 12 3e 0d  |>30<oht.^-....>.|
00000260  46 3d 68 74 74 10 5f 74  68 70 65 61 64 33 3e 32  |F=htt._thpead3>2|
00000270  3c 2f 68 74 74 70 5f 74  68 72 65 61 24 73 3e 7e  |</http_threa$s>~|
00000280  2f 68 74 74 70 3e 0a 3c  73 79 6e 3e 3c 73 79 6e  |/http>.<syn><syn|
00000290  5f 66 72 65 71 3e 32 30  3c 2f 73 79 6e 5f 66 72  |_freq>20</syn_fr|
```

- So, this looks like there’s a pair of tags like “ `<http>` ” stuff “ `</http>` ” and more “ `http_freq` ” and “ `http_threads` ” stuff…

  “ `<Nttp$<http_freq>30<oht\xB4^-\x03\x1E\x0A\x12\x3E\x0D` “

  “ `F=htt\x10_thpead3>2</http_threa$s>~/http>x\0A` “

- So that mostly becomes:

  “ `<http><http_freq>30</http_freq>F<http_threads>2</http_threads></http>` “

All the rest of the file looks fine… so near the beginning where it says:

```
000000e0  3e 0a 3c 63 6d 64 73 3e  5a 79 2f 63 21 65 77 3e  |>.<cmds>Zy/c!ew>|
000000f0  73 7e c3 24 67 5f 64 61  74 61 3e 0a dc 64 6a 4e  |s~.$g_data>..djN|
00000100  78 3f 3b 7e 63 7e 5f 73  69 40 65 3e 31 30 30 30  |x?;~c~_si@e>1000|
```

- “ `<cmds>Zy/c!ew>s~\xC3$g_data>\x0A` ” based upon the rest of the file, that should probably be:

  “ `<cmds>Z</cmds>s<plg_data>` ” the EOL character used after the closing tag(s) of the larger structures is “ `\x0A` ” rather than a “ `\x0D\x0A` ” after each config option.
- So, the “ `s` ” is probably a “ `\x0A` “
- If this is also true of all opening tags, then the “ `Z` ” in commands should be a “ `\x0A` ” also.

- So, do all of the tags balance? There’s a closing “ `</ddos>` ” tag without an opening tag. If this XML document is well formed, the opening tag should go here:

  “ `<plg_data>\x0A\xDCdjNx?<tcp_size>` ” between the opening “ `<plg_data>` ” and “ `<tcp_size>` ” tags.
- “ `<plg_data>\x0A<ddos><tcp_size>` ” It fits just right.

(If you were paying close attention, you may have noticed that many of the corrected bytes, differ from the original by only a single bit.)

Recap

So let’s see what we have so far:

```
00000000  3c 3f 78 6d 6c 20 76 65  72 73 69 6f 6e 3d 22 31  |<?xml version="1|
00000010  2e 30 22 3f 3e 0a 3c 62  6b 65 72 6e 65 6c 3e 0a  |.0"?>.<bkernel>.|
00000020  3c 70 6c 75 67 69 6e 73  3e 0a 3c 70 6c 75 67 69  |<plugins>.<plugi|
00000030  6e 3e 0a 3c 6e 61 6d 65  3e 64 64 6f 73 3c 2f 6e  |n>.<name>ddos</n|
00000040  61 6d 65 3e 0a 88 7f a8  53 cb 68 23 a3 1f 65 54  |ame>....S.h#..eT|
00000050  46 05 45 02 01 06 08 1c  5f 67 1c 4c 11 02 1b 08  |F.E....._g.L....|
00000060  1d 4e 5c 6f 1c 02 19 1b  47 00 00 1e 4e 73 3d 41  |.N\o....G...Ns=A|
00000070  00 0a 5a 0d 5a 79 7d 36  0b 6e 61 6d 65 3e 0a 3c  |..Z.Zy}6.name>.<|
00000080  11 0a be 86 4a 61 cc 98  12 32 8d d0 46 7c d1 cf  |....Ja...2..F|..|
00000090  49 6c c1 ac 1d 21 d2 ca  95 66 96 c8 1f 04 9e d6  |Il...!...f......|
000000a0  5c 73 98 cf 4f 30 a8 9a  ce 67 92 c3 16 7d db c8  |\s..O0...g...}..|
000000b0  1f 21 cd c7 68 6b 9c ac  1a 74 a7 d4 57 67 cd c8  |.!..hk...t..Wg..|
000000c0  18 33 c2 89 54 6b d0 d5  4f 6d 96 98 28 32 8d d6  |.3..Tk..Om..(2..|
000000d0  3e 1c 04 01 4d 30 a8 3c  2f 70 6c 75 67 69 6e 73  |>...M0.</plugins|
000000e0  3e 0a 3c 63 6d 64 73 3e  0a 3c 2f 63 6d 64 73 3e  |>.<cmds>.</cmds>|
000000f0  0a 3c 70 6c 67 5f 64 61  74 61 3e 0a 3c 64 64 6f  |.<plg_data>.<ddo|
00000100  73 3e 3c 74 63 70 5f 73  69 7a 65 3e 31 30 30 30  |s><tcp_size>1000|
00000110  3c 2f 74 63 70 5f 73 69  7a 65 3e 0d 0a 3c 74 63  |</tcp_size>..<tc|
00000120  70 5f 66 72 65 71 3e 35  30 3c 2f 74 63 70 5f 66  |p_freq>50</tcp_f|
00000130  72 65 71 3e 0d 0a 3c 74  63 70 5f 74 68 72 65 61  |req>..<tcp_threa|
00000140  64 73 3e 35 3c 2f 74 63  70 5f 74 68 72 65 61 64  |ds>5</tcp_thread|
00000150  73 3e 0d 0a 3c 75 64 70  5f 73 69 7a 65 3e 31 30  |s>..<udp_size>10|
00000160  30 30 3c 2f 75 64 70 5f  73 69 7a 65 3e 0d 0a 3c  |00</udp_size>..<|
00000170  75 64 70 5f 66 72 65 71  3e 35 30 3c 2f 75 64 70  |udp_freq>50</udp|
00000180  5f 66 72 65 71 3e 0d 0a  3c 75 64 70 5f 74 68 72  |_freq>..<udp_thr|
00000190  65 61 64 73 3e 35 3c 2f  75 64 70 5f 74 68 72 65  |eads>5</udp_thre|
000001a0  61 64 73 3e 0d 0a 3c 69  63 6d 70 5f 73 69 7a 65  |ads>..<icmp_size|
000001b0  3e 31 30 30 30 3c 2f 69  63 6d 70 5f 73 69 7a 65  |>1000</icmp_size|
000001c0  3e 0d 0a 3c 69 63 6d 70  5f 66 72 65 71 3e 35 30  |>..<icmp_freq>50|
000001d0  3c 2f 69 63 6d 70 5f 66  72 65 71 3e 0d 0a 3c 69  |</icmp_freq>..<i|
000001e0  63 6d 70 5f 74 68 72 65  61 64 73 3e 35 3c 2f 69  |cmp_threads>5</i|
000001f0  63 6d 70 5f 74 68 72 65  61 64 73 3e 0d 0a 3c 68  |cmp_threads>..<h|
00000200  74 74 70 5f 66 72 65 71  3e 43 54 51 3c 2f 68 74  |ttp_freq>CTQ</ht|
00000210  74 70 5f 66 72 65 71 3e  0d 0a 3c 68 74 74 70 5f  |tp_freq>..<http_|
00000220  74 68 72 65 61 64 73 3e  35 3c 2f 68 74 74 70 5f  |threads>5</http_|
00000230  74 68 72 65 61 64 73 3e  3c 2f 64 64 6f 73 3e 0a  |threads></ddos>.|
00000240  3c 68 74 74 70 3e 3c 68  74 74 70 5f 66 72 65 71  |<http><http_freq|
00000250  3e 33 30 3c 2f 68 74 74  70 5f 66 72 65 71 3e 0d  |>30</http_freq>.|
00000260  0a 3c 68 74 74 70 5f 74  68 72 65 61 64 73 3e 32  |.<http_threads>2|
00000270  3c 2f 68 74 74 70 5f 74  68 72 65 61 64 73 3e 3c  |</http_threads><|
00000280  2f 68 74 74 70 3e 0a 3c  73 79 6e 3e 3c 73 79 6e  |/http>.<syn><syn|
00000290  5f 66 72 65 71 3e 32 30  3c 2f 73 79 6e 5f 66 72  |_freq>20</syn_fr|
000002a0  65 71 3e 0d 0a 3c 73 79  6e 5f 74 68 72 65 61 64  |eq>..<syn_thread|
000002b0  73 3e 33 3c 2f 73 79 6e  5f 74 68 72 65 61 64 73  |s>3</syn_threads|
000002c0  3e 3c 2f 73 79 6e 3e 0a  3c 2f 70 6c 67 5f 64 61  |></syn>.</plg_da|
000002d0  74 61 3e 0a 3c 73 6c 65  65 70 66 72 65 71 3e 39  |ta>.<sleepfreq>9|
000002e0  30 30 3c 2f 73 6c 65 65  70 66 72 65 71 3e 0a 3c  |00</sleepfreq>.<|
000002f0  69 70 3e 31 32 38 2e 31  33 30 2e 35 36 2e 32 32  |ip>128.130.56.22|
00000300  3c 2f 69 70 3e 0a 3c 2f  62 6b 65 72 6e 65 6c 3e  |</ip>.</bkernel>|
00000310  0a                                                |.|
00000311
```

Unknown Bits

There’s still a chunk of noise near the beginning.

From what we can see, there must be a “ `</plugin>` ” tag, and an opening or closing “ `name` ” tag, somewhere in the noise. Is there anything in the other combinations of files? So… the answer is no, not really, all three combinations of these files have
identical noise between offsets `0x40` and `0x70`, and unique noise from `0x80` to `0xd0`.
So whatever is between `0x40` and `0x70` is common to all these EXEs.

```
00000040  6f 72 df 30 0a 88 7f a8  53 cb 68 23 a3 1f 65 54  |or.0....S.h#..eT|
00000050  46 05 45 02 01 06 08 1c  5f 67 1c 4c 11 02 1b 08  |F.E....._g.L....|
00000060  1d 4e 5c 6f 1c 02 19 1b  47 00 00 1e 4e 73 3d 41  |.N\o....G...Ns=A|
00000070  00 0a 5a 0d 5a 79 7d 36  0b 6e 61 6d 65 3e 0a 3c  |..Z.Zy}6.name>.<|
00000080  9b db b5 e7 c0 b0 c7 f9  98 e3 86 b1 cc ad da ae  |................|
00000090  c3 bd 98 cd 96 f0 d9 ab  d9 b4 9f a9 96 d5 95 b7  |................|
000000a0  c0 a6 93 ae c5 e1 a3 fb  44 b6 9b a2 96 ac d0 a9  |........D.......|
000000b0  16 f8 9a a6 c3 ba 97 cd  95 a9 cd b5 c9 b6 c6 a9  |................|
000000c0  92 e2 f5 e8 da ba db b4  c5 bc 9b f9 a2 e3 86 b7  |................|
000000d0  c0 a6 94 ae c6 e1 a3 fb  7d 19 0f 1d ce b6 c7 b4  |........}.......|
```

Fixup the EXE

So, let’s extract as much of the keystream as we have now, and take a look at the syn.exe file…

```
perl xor3.pl xml.crypt demo demo.keystream
perl xor3.pl syn.crypt demo.keystream demo.syn
hexdump -C demo.syn
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 e8 00 00 00  |................|
00000040  0e 1f ba 0e 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000000d0  00 00 00 00 00 00 00 a6  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
000000f0  79 42 b3 48 00 00 00 00  00 00 00 00 e0 00 0e 21  |yB.H...........!|
00000100  0b 01 07 0a 00 0e 00 00  00 3a 00 00 00 00 00 00  |.........:......|
00000110  ac 10 00 00 00 10 00 00  00 20 00 00 00 00 00 10  |......... ......|
00000120  00 10 00 00 00 02 00 00  04 00 00 00 00 00 00 00  |................|
00000130  04 00 00 00 00 00 00 00  00 70 00 00 00 04 00 00  |.........p......|
00000140  00 00 00 00 02 00 00 04  00 00 10 00 00 10 00 00  |................|
00000150  00 00 10 00 00 10 00 00  00 00 00 00 10 00 00 00  |................|
00000160  10 26 00 00 62 00 00 00  08 22 00 00 b4 00 00 00  |.&..b...."......|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  00 00 00 00 00 00 00 00  00 60 00 00 bc 00 00 00  |.........`......|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001c0  00 20 00 00 b4 00 00 00  00 00 00 00 00 00 00 00  |. ..............|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  2e 74 65 78 74 00 00 00  00 0d 00 00 00 10 00 00  |.text...........|
000001f0  00 0e 00 00 00 04 00 00  00 00 00 00 00 00 00 00  |................|
00000200  00 00 00 00 20 00 00 60  2e 00 00 00 74 61 00 00  |.... ..`....ta..|
00000210  72 06 00 00 00 20 00 00  00 08 00 00 00 12 00 00  |r.... ..........|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 40  |............@..@|
00000230  2e 64 61 74 61 00 00 00  70 2e 00 00 00 30 00 00  |.data...p....0..|
00000240  00 26 00 00 00 1a 00 00  00 00 00 00 00 00 00 00  |.&..............|
00000250  00 00 00 00 40 00 00 c0  2e 72 65 6c 6f 63 00 00  |....@....reloc..|
00000260  4c 01 00 00 00 60 00 00  00 02 00 00 00 40 00 00  |L....`.......@..|
00000270  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 42  |............@..B|
00000280  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000310  00                                                |.|
00000311
```

Well this is encouraging…

The section header names all look pretty standard, so
“ `.\x00\x00\x00ta` ” is probably “ `.rdata` ” or “ `.idata` ” (maybe even `pdata`)
This corresponds with the mystery argument to “ `http_freq` ” in the config file.

```
00000200  74 74 70 5f 66 72 65 71  3e 43 54 51 3c 2f 68 74  |ttp_freq>CTQ</ht|
00000200  00 00 00 00 20 00 00 60  2e 00 00 00 74 61 00 00  |.... ..`....ta..|
```

- XOR “ `CTQ` ” by “ `ida` ” and you get “ `*00` “
- XOR “ `CTQ` ” by “ `rda` ” and you get “ `100` “
- XOR “ `CTQ` ” by “ `pda` ” and you get “ `300` “
- But, just like with the config file above; We can XOR it against the other EXEs

```
http.crypt ⊕ syn.crypt
00000200  00 0e 00 00 20 04 00 60  2e 72 64 61 74 61 00 00  |.... ...rdata..|
00000210  72 06 00 00 20 20 00 60  2e 7a 64 61 74 73 00 00  |r...  ..zdats..|
```

> The answer is ‘ `rdata` ‘.

You can also edit this EXE in your favorite 8-bit clean text editor, or your favorite hex editor if you’re not a massochist.

DOS Header

The “ `PE\x00\x00` ” header is at offset `0xE8` and so the DWORD at `0x3C` is correct.

For review, this is the structure of a DOS EXE header:

```
 typedef struct _IMAGE_DOS_HEADER {  // DOS .EXE header
USHORT e_magic;         // Magic number
USHORT e_cblp;          // Bytes on last page of file
USHORT e_cp;            // Pages in file
USHORT e_crlc;          // Relocations
USHORT e_cparhdr;       // Size of header in paragraphs
USHORT e_minalloc;      // Minimum extra paragraphs needed
USHORT e_maxalloc;      // Maximum extra paragraphs needed
USHORT e_ss;            // Initial (relative) SS value
USHORT e_sp;            // Initial SP value
USHORT e_csum;          // Checksum
USHORT e_ip;            // Initial IP value
USHORT e_cs;            // Initial (relative) CS value
USHORT e_lfarlc;        // File address of relocation table
USHORT e_ovno;          // Overlay number
USHORT e_res[4];        // Reserved words
USHORT e_oemid;         // OEM identifier (for e_oeminfo)
USHORT e_oeminfo;       // OEM information; e_oemid specific
USHORT e_res2[10];      // Reserved words
LONG   e_lfanew;        // File address of new exe header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
... followed immediately by the stub program.
```

http://www.csn.ul.ie/~caolan/publink/winresdump/winresdump/doc/pefile.html (or *www.skynet.ie*)

The usual stuff *http://msdn.microsoft.com/en-us/magazine/cc301805.aspx*
Since everything is a WORD, let’s alter our hexdump slightly:

```
0000000 5a4d 0090 0003 0000 0004 0000 ffff 0000 e_magic; e_cblp;    e_cp;    e_crlc; e_cparhdr; e_minalloc;
e_maxalloc; e_ss;
0000010 00b8 0000 0000 0000 0040 0000 0000 0000 e_sp;    e_csum;    e_ip;      e_cs;  e_lfarlc;     e_ovno;
e_res  e_res
0000020 0000 0000 0000 0000 0000 0000 0000 0000 e_res     e_res  e_oemid; e_oeminfo;    e_res2      e_res2
e_res2  e_res2
0000030 0000 0000 0000 0000 0000 0000 00e8 0000 e_res2   e_res2   e_res2     e_res2     e_res2      e_res2
e_lfanew..e_lfanew;
0000040 1f0e 0eba 0000 0000 0000 0000 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
```

And make it a bit easier to read just for this blog post:

Offset	Name		Value	
0x00	Signature	e_magic	0x5a4d	“MZ”
0x02	Byte on Last Page	e_cblp	0x0090	144.0 bytes
0x04	Page Count	e_cp	0x0003	3.0 pages
0x06	Relocations Count	c_crlc	0x0000	0.0
0x08	Header Size	e_cparhdr	0x0004	4.0 paragraphs
0x0A	Minimum Memory	e_minalloc	0x0000	0.0 bytes
0x0C	Maximum Memory	e_maxalloc	0xffff	65535.0 bytes
0x0E	SS : SP	e_ss..e_sp	0x0000 0x00b8	0000:00B8h
0x12	Checksum	e_csum	0x0000	0.0
0x14	CS : IP	e_ip..e_cs	0x0000 0x0000	0000:0000h
0x18	Relocation Table Offset	e_lfarlc	0x0040	0×40 bytes
0x1A	Overlay Number	e_ovno	0x0000	
	(Entry Point)			(00000040h)

`0x1C`	Reserved	`e_res..e_res`	`0x0000 0x0000 0x0000 0x0000`	0.0
`0x24`	Module Length/OEM Identifier	`e_oemid`	`0x0000`	0.0
`0x26`	Image Offset/OEM Information	`e_oeminfo`	`0x0000`	0.0
`0x28`	Reserved	`e_res2..e_res2`	`0x0000` times 10	0.0
`0x3C`	New EXE [PE] Header Offset	`e_lfanew..e_lfanew`	`0x00e8 0x0000`	000000E8h

So let's assume for the moment, that offsets `0x00` thru `0x3F` are correct, as they look correct in the recovered XML file. Everything from `0xD7` to the end looks mostly correct too, since this is mostly a PE header, it should be possible to sanity check this later.

The PE headers between the `http.exe` and `syn.exe` are shifted by (`0x10`) 16 bytes, so… `syn.exe` is the one `0x10` bytes ahead. (The top lines of each pair.)

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000030  00 00 00 00 00 00 00 00  00 00 00 00 10 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000080  8a d1 0b 61 8a d1 0b 61  8a d1 0b 61 8a d1 0b 61  |...a...a...a...a|
00000090  8a d1 59 61 8b d1 0b 61  4c d2 09 61 89 d1 0b 61  |..Ya...aL..a...a|
000000a0  9c d5 0b 61 8a d1 0b 61  8a d1 09 61 80 d1 0b 61  |...a...a...a...a|
000000b0  09 d9 57 61 ab d1 0b 61  8f dd 6a 61 9e d1 0b 61  |..Wa...a..ja...a|
000000c0  8a d1 37 61 8e d1 0b 61  8a d1 0d 61 8a d1 0b 61  |..7a...a...a...a|
000000d0  fe ba 90 af 8b d1 0b 61  52 69 63 68 a9 df a9 c7  |.......aRich....|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
000000f0  79 42 b3 48 00 00 00 00  50 45 00 00 ac 01 0a 21  |yB.H....PE.....!|
00000100  9e 27 a1 42 00 0e 00 00  00 3a 00 00 e0 00 0e 21  |.'.B.....:.....!|
00000110  a7 11 07 0a 00 1e 00 00  00 30 00 00 00 00 00 10  |.........0......|
00000120  41 00 00 00 00 12 00 00  04 20 00 00 00 00 00 10  |A........ ......|
00000130  04 10 00 00 00 02 00 00  04 70 00 00 00 04 00 00  |.........p......|
00000140  04 00 00 00 02 00 00 04  00 50 10 00 00 14 00 00  |.........P......|
00000150  00 00 10 00 02 10 00 04  00 00 10 00 10 10 00 00  |................|
00000160  10 26 10 00 62 10 00 00  08 22 00 00 a4 00 00 00  |.&..b...."......|
00000170  10 23 00 00 63 00 00 00  00 21 00 00 64 00 00 00  |.#..c....!..d...|
00000180  00 00 00 00 00 00 00 00  00 60 00 00 bc 00 00 00  |.........`......|
00000190  00 00 00 00 00 00 00 00  00 40 00 00 20 01 00 00  |.........@.. ...|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001c0  00 20 00 00 b4 00 00 00  00 00 00 00 00 00 00 00  |. ..............|
000001d0  00 20 00 00 5c 00 00 00  00 00 00 00 00 00 00 00  |. ..\...........|
000001e0  2e 74 65 78 74 00 00 00  00 0d 00 00 00 10 00 00  |.text...........|
000001f0  2e 7a 65 78 74 04 00 00  dd 0d 00 00 00 10 00 00  |.zext...........|
00000200  00 0e 00 00 20 04 00 60  2e 72 64 61 74 61 00 00  |.... ..`.rdata..|
00000210  72 06 00 00 20 20 00 60  2e 7a 64 61 74 73 00 00  |r...  .`.zdats..|
00000220  73 03 00 00 00 20 00 00  00 04 00 00 40 12 00 40  |s.... ......@..@|
00000230  2e 64 61 74 61 00 00 00  70 2e 00 00 40 30 00 40  |.data...p...@0.@|
00000240  2e 42 61 74 61 1a 00 00  f0 09 00 00 00 30 00 00  |.Bata........0..|
00000250  00 02 00 00 40 16 00 c0  2e 72 65 6c 6f 63 00 00  |....@....reloc..|
00000260  4c 01 00 00 40 60 00 c0  2e 70 65 6c 6f 23 00 00  |L...@`...pelo#..|
00000270  86 01 00 00 00 40 00 00  00 02 00 00 40 18 00 42  |.....@......@..B|
00000280  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 42  |............@..B|
00000290  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Bytes between 0×40 and 0x7F are the same in all files, however,

1. If our XML was right, the first four bytes are `0e 1f ba 0e`

   ```
   00000040  0e 1f ba 0e 00 00 00 00  00 00 00 00 00 00 00 00  |................|
   ```

2. These are fairly standard EXE headers, and these `0x40` bytes come right after the DOS EXE header… The entry point of this DOS executable is located at offset `0x40` too.

   So what usually follows the DOS EXE header, and is `0x40` bytes long?

   http://www.google.com/search?q=dos+exe+stub

```
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000040  0E                 push cs
00000041  1F                 pop ds
00000042  BA0E00             mov dx,0xe    ; string offset "This program…"
00000045  B409               mov ah,0x9
00000047  CD21               int 0x21      ; print
00000049  B8014C             mov ax,0x4c01
0000004C  CD21               int 0x21      ; exit
0000004E  db "This program cannot be run in DOS mode.\r\r\n$",0
```

There are several variations on the DOS stub. This is one of the more likely ones, as it's the default Microsoft one.
Also note where the Nulls match up with the config file:

```
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000070  00 0a 5a 0d 5a 79 7d 36  0b 6e 61 6d 65 3e 0a 3c  |..Z.Zy}6.name>.<|
```

So, let's XOR the stub (from `0x45` to `0x78`) with the config file:

```
00000000  3c 3f 78 6d 6c 20 76 65  72 73 69 6f 6e 3d 22 31  |<?xml version="1|
00000010  2e 30 22 3f 3e 0a 3c 62  6b 65 72 6e 65 6c 3e 0a  |.0"?>.<bkernel>.|
00000020  3c 70 6c 75 67 69 6e 73  3e 0a 3c 70 6c 75 67 69  |<plugins>.<plugi|
00000030  6e 3e 0a 3c 6e 61 6d 65  3e 64 64 6f 73 3c 2f 6e  |n>.<name>ddos</n|
00000040  61 6d 65 3e 0a 3c 76 65  72 73 69 6f 6e 3e 31 3c  |ame>.<version>1<|
00000050  2f 76 65 72 73 69 6f 6e  3e 0a 3c 2f 70 6c 75 67  |/version>.</plug|
00000060  69 6e 3e 0a 3c 70 6c 75  67 69 6e 3e 0a 3c 6e 61  |in>.<plugin>.<na|
00000070  6d 65 3e 68 74 74 70 3c  2f 6e 61 6d 65 3e 0a 3c  |me>http</name>.<|
00000080  11 0a be 86 4a 61 cc 98  12 32 8d d0 46 7c d1 cf  |....Ja...2..F|..|
00000090  49 6c c1 ac 1d 21 d2 ca  95 66 96 c8 1f 04 9e d6  |Il...!...f......|
000000a0  5c 73 98 cf 4f 30 a8 9a  ce 67 92 c3 16 7d db c8  |\s..O0...g...}..|
000000b0  1f 21 cd c7 68 6b 9c ac  1a 74 a7 d4 57 67 cd c8  |.!..hk...t..Wg..|
000000c0  18 33 c2 89 54 6b d0 d5  4f 6d 96 98 28 32 8d d6  |.3..Tk..Om..(2..|
000000d0  3e 1c 04 01 4d 30 a8 3c  2f 70 6c 75 67 69 6e 73  |>...M0.</plugins|
000000e0  3e 0a 3c 63 6d 64 73 3e  0a 3c 2f 63 6d 64 73 3e  |>.<cmds>.</cmds>|
000000f0  0a 3c 70 6c 67 5f 64 61  74 61 3e 0a 3c 64 64 6f  |.<plg_data>.<ddo|
00000100  73 3e 3c 74 63 70 5f 73  69 7a 65 3e 31 30 30 30  |s><tcp_size>1000|
```

There are those missing XML tags. It would not be unreasonable to guess that in that noise is also a " `<name>ddos</name>` " and " `<name>syn</name>` "

Almost there, now there's just the bytes between `0x80` and `0xD7` left to go.

```
perl /home/jwolf/xor3.pl xml.crypt demo2 demo2.keystream
DEBUG: xml.crypt ^ demo2 -> demo2.keystream
perl /home/jwolf/xor3.pl syn.crypt demo2.keystream demo2.syn
DEBUG: syn.crypt ^ demo2.keystream -> demo2.syn
```

Rich Header

So, what about the bytes between `0x80` and `0xD7` ? In recent versions of Visual C++ (2003 and onwards I think), the linker has been
putting an extra chunk into the EXE headers between the DOS stub, and the PE headers. The " `Rich` " in the `http.exe⊕syn.exe` dump above rather gives it away.

Using a time machine to skip to the end of this blog post, these are the decrypted headers from the " `syn.exe` " file. I'll be using it for the examples below.

```
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  67 6f cc f5 23 0e a2 a6  23 0e a2 a6 23 0e a2 a6  |go..#...#...#...|
00000090  26 02 ff a6 21 0e a2 a6  e0 01 ff a6 21 0e a2 a6  |&...!.......!...|
000000a0  30 06 ff a6 21 0e a2 a6  a0 06 ff a6 28 0e a2 a6  |0...!.......(...|
000000b0  23 0e a3 a6 05 0e a2 a6  26 02 c2 a6 24 0e a2 a6  |#.......&...$...|
000000c0  26 02 fe a6 22 0e a2 a6  26 02 f8 a6 22 0e a2 a6  |&..."...&..."...|
000000d0  52 69 63 68 23 0e a2 a6  00 00 00 00 00 00 00 00  |Rich#...........|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
```

Note how the `NULL`s lined up:

```
000000d0  52 69 63 68 23 0e a2 a6  00 00 00 00 00 00 00 00  |Rich#...........| future decrypted syn.exe
000000d0  ac d3 f3 c7 a8 df a9 c7  52 69 63 68 a9 df a9 c7  |........Rich....| future decrypted http.exe
000000d0  fe ba 90 af 8b d1 0b 61  52 69 63 68 a9 df a9 c7  |.......aRich....| syn.exe⊕http.exe
```

Rich Header Structure

A hash is calculated by adding together each byte of the EXE file up to the beginning of the Rich header, with each byte being shifted left by the number of bits that is its file offset. Take a look at the code below if that doesn't make sense. For the " `syn.exe` " file, the `0x80` bytes of the DOS header and stub come out to `0x884f3421`.

This partial total, then has each DWORD of the list inside the (unencrypted) Rich header, shifted left by the other DWORD written after it (though it always seems to be under a byte in length. Values over 32 don't make sense.) This number — the final total — is the hash.

The first 16 bytes of the header are this hash `XOR`'d with this string:

```
00000080  44 61 6e 53 00 00 00 00  00 00 00 00 00 00 00 00  |DanS............|
```

Using the " `syn.exe` " file again as an example, the hash total has was `0xa6a20e23` When `XOR`'d by this value, those 16 bytes become:

```
00000080  67 6f cc f5 23 0e a2 a6  23 0e a2 a6 23 0e a2 a6  |go..#...#...#...|
```

Then the version number of each library used in compiling and linking this EXE are written as DWORDs followed by the shift used for the earlier hash calculation. This section will always be a multiple of eight bytes in length. For example:

```
            Version      Shift       Version      Shift
00000090  05 0c 5d 00 02 00 00 00  c3 0f 5d 00 02 00 00 00  |..].......].....|
000000a0  13 08 5d 00 02 00 00 00  83 08 5d 00 0b 00 00 00  |..].......].....|
000000b0  00 00 01 00 26 00 00 00  05 0c 60 00 07 00 00 00  |....&.....`.....|
000000c0  05 0c 5c 00 01 00 00 00  05 0c 5a 00 01 00 00 00  |..\.......Z.....|
```

Then, at the end of the list, there is the constant value "Rich" (0x68636952), followed by the hash, and then there is either 8, 16, or 24 bytes of NULLs after, depending on what the value of the hash was. None of this is XOR'd. So it looks like this:

```
                         Hash      sixteen bytes of padding
000000d0  52 69 63 68 23 0e a2 a6  00 00 00 00 00 00 00 00  |Rich#...........|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
```

Something to take note of is the large number of NULLs in this structure, if you're going to be using this structure to perform known plaintext attacks. In the EXE, each of these will be a byte of the hash value. Also, there are only a small number of possible version numbers.

```
00000080  44 61 6e 53 00 00 00 00  00 00 00 00 00 00 00 00  |DanS............|
00000090  05 0c 5d 00 02 00 00 00  c3 0f 5d 00 02 00 00 00  |..].......].....|
000000a0  13 08 5d 00 02 00 00 00  83 08 5d 00 0b 00 00 00  |..].......].....|
000000b0  00 00 01 00 26 00 00 00  05 0c 60 00 07 00 00 00  |....&.....`.....|
000000c0  05 0c 5c 00 01 00 00 00  05 0c 5a 00 01 00 00 00  |..\.......Z.....|
```

@comp.id

Nobody (publicly) knows exactly what is in the @comp.id. The low 16 bits is quite clearly the build number in the version string of the COFF object that was linked in. And some people think that the high 16 bits are the major and minor version numbers of the same. But that's doesn't seem to work. I've collected a short list of some @coff.id symbols that I could find. The same compiler will output two different @coff.id's depending on things like whether it's compiling C or C++. And mathematically, I can't figure out the exact transform to turn the top 16 bits into the correct major and minor version. For example, in "syn.dll", the first @comp.id in the Rich header is 0x005d0c05. From the PE Optional header, we can confirm that this dll was compiled with Visual C++ "7.10". The 0x0c05 part is clearly "3077", but the 0x005d part doesn't quite turn into "7.10". The "0xD" is 13, and "0x5" is 5. So, that would make "13.5.3077". That version of Visual Studio is "13" I think, so that could be right. And we could multiply 5 by 2 to get 10. But then this doesn't work for other example @comp.id's.

I put together a short table of all the @comp.id's that I could find in an hour or two. Please add to it. It's in Appendix E. (I'm going to ask some people at Microsoft about this soon.)

Back to the present

Let's pretend that you didn't know the hash.

It apparently ends with 0xA6 [the most significant byte], from what's already known of the keystream, vis:

```
000000d0  00 00 00 00 00 00 00 a6  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
```

And we can XOR the "Syn" EXE with the "Http" EXE. The RICH headers of one are 8 bytes longer than the other. We also know what the first sixteen bytes of both, is the XOR of the string "DanS" with both hashes. Both "DanS"'s would have canceled out, so the XOR of the two hashes is **0x610BD18A** repeated four times.

```
00000080  8a d1 0b 61 8a d1 0b 61  8a d1 0b 61 8a d1 0b 61  |...a...a...a...a|
00000090  8a d1 59 61 8b d1 0b 61  4c d2 09 61 89 d1 0b 61  |..Ya...aL..a...a|
000000a0  9c d5 0b 61 8a d1 0b 61  8a d1 09 61 80 d1 0b 61  |...a...a...a...a|
000000b0  09 d9 57 61 ab d1 0b 61  8f dd 6a 61 9e d1 0b 61  |..Wa...a..ja...a|
000000c0  8a d1 37 61 8e d1 0b 61  8a d1 0d 61 8a d1 0b 61  |..7a...a...a...a|
000000d0  fe ba 90 af 8b d1 0b 61  52 69 63 68 a9 df a9 c7  |.......aRich....|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...|
```

We can also see a visible "Rich" header above — from the "Http" EXE. If the "Syn" EXE didn't have anything at that same offset, the hash of the "Http" EXE is **0xC7A9DFA9**

So, 0x610BD18A XOR 0xC7A9DFA9 = 0xA6A20E23, which is ultimately the correct hash for the "Syn" EXE.

Alternatively

Since this is how I actually did it the first time, without knowing as much as I do now about the Rich header, and I want to show the work off…

I XOR'd the two hashes which come right after the "Rich".

```
000000d0  fe ba 90 af 8b d1 0b 61  52 69 63 68 a9 df a9 c7  |.......aRich....|
```

That's 8b d1 0b 61 XOR a9 df a9 c7 = 22 0e a2 a6 (0xA6A20E22), which is off by one — the one bit being an occurrence count of "0x00000001" for whichever @comp.id that was. (But pretend that we don't know that yet.)

So, I XOR'd 0xA6A20E22 with the XML data between offsets 0x80 and 0xd7, and look! Almost done!

```
00000000  3c 3f 78 6d 6c 20 76 65  72 73 69 6f 6e 3d 22 31  |<?xml version="1|
00000010  2e 30 22 3f 3e 0a 3c 62  6b 65 72 6e 65 6c 3e 0a  |.0"?>.<bkernel>.|
00000020  3c 70 6c 75 67 69 6e 73  3e 0a 3c 70 6c 75 67 69  |<plugins>.<plugi|
00000030  6e 3e 0a 3c 6e 61 6d 65  3e 64 64 6f 73 3c 2f 6e  |n>.<name>ddos</n|
00000040  61 6d 65 3e 0a 3c 76 65  72 73 69 6f 6e 3e 31 3c  |ame>.<version>1<|
00000050  2f 76 65 72 73 69 6f 6e  3e 0a 3c 2f 70 6c 75 67  |/version>.</plug|
00000060  69 6e 3e 0a 3c 70 6c 75  67 69 6e 3e 0a 3c 6e 61  |in>.<plugin>.<na|
00000070  6d 65 3e 68 74 74 70 3c  2f 6e 61 6d 65 3e 0a 3c  |me>http</name>.<|
00000080  33 04 1c 20 68 6f 6e 3e  30 3c 2f 76 64 72 73 69  |3.. hon>0</vdrsi|
00000090  6b 62 63 0a 3f 2f 70 6c  b7 68 34 6e 3d 0a 3c 70  |kbc.?/pl.h4n=.<p|
000000a0  7e 7d 3a 69 6d 3e 0a 3c  ec 69 30 65 34 73 79 6e  |~}:im>.<.i0e4syn|
000000b0  3d 2f 6f 61 4a 65 3e 0a  38 7a 05 72 75 69 6f 6e  |=/oaJe>.8z.ruion|
000000c0  3a 3d 60 2f 76 65 72 73  6d 63 34 3e 0a 3c 2f 70  |:=`/versmc4>.</p|
000000d0  1c 12 a6 a7 6f 3e 0a 9a  2f 70 6c 75 67 69 6e 73  |....o>../plugins|
000000e0  3e 0a 3c 63 6d 64 73 3e  0a 3c 2f 63 6d 64 73 3e  |>.<cmds>.</cmds>|
000000f0  0a 3c 70 6c 67 5f 64 61  74 61 3e 0a 3c 64 64 6f  |.<plg_data>.<ddo|
00000100  73 3e 3c 74 63 70 5f 73  69 7a 65 3e 31 30 30 30  |s><tcp_size>1000|
00000110  3c 2f 74 63 70 5f 73 69  7a 65 3e 0d 0a 3c 74 63  |</tcp_size>..<tc|
```

Note

Although not used here, don't forget you can XOR out the "Rich" and "DanS" DWORDS where appropriate.

Fixing the last chunk of XML

So pull this back up in a text editor and make another correction pass over it. Since half the DWORDS are less than 0x00000010, expect to see a run of three good characters, followed by a bad one. And every @comp.id is 0x00 for the high byte, expect three garbled characters followed by one good one. (And all these offsets DWORD aligned.) [And I know I basically said the same thing above. I wrote post in small chunks over two months, and I don't want to do more editing than I need to.]

```
I'm doing these example with the XOR hash that's slightly off. Since I already wrote up all the hexdumps and
correction prose.
```

```
00000080  33 04 1c 20 68 6f 6e 3e  30 3c 2f 76 64 72 73 69  |3.. hon>0</vdrsi|
00000090  6b 62 63 0a 3f 2f 70 6c  b7 68 34 6e 3d 0a 3c 70  |kbc.?/pl.h4n=.<p|
```

- " `<3\x04\x1c hon>0</vdrsikbc\x0A` " I believe looks like:
  " `<version>0</version>\x0A` "
- The the next byte is garbled, but " `/pl` " should be correct, and the next three garbled, but " `n` " and then " `\x0A<pl` " are correct.

  This is probably the closing "plugin" tag.

  " `?/pl\xB7h4n=\x0A<p` " becomes " `</plugin>\x0A<p` "
- " `<p~}:im>\x0A<" the "~}:` " is wrong, the " `i` " should be correct, and the " `m` " wrong, while " `>\x0A<` " is correct.
- I'm going to guess this also says "plugin"

  " `<plugin>\x0A<` "

```
000000a0  7e 7d 3a 69 6d 3e 0a 3c  ec 69 30 65 34 73 79 6e  |~}:im>.<.i0e4syn|
000000b0  3d 2f 6f 61 4a 65 3e 0a  38 7a 05 72 75 69 6f 6e  |=/oaJe>.8z.ruion|
000000c0  3a 3d 60 2f 76 65 72 73  6d 63 34 3e 0a 3c 2f 70  |:=`/versmc4>.</p|
000000d0  1c 12 a6 a7 6f 3e 0a 9a  2f 70 6c 75 67 69 6e 73  |....o>../plugins|
```

- " `<\xECi0e4syn` " the " `\xECi0` " part is garbled, the " `e` " is correct, the " `4` " isn't, and the " `syn` " is correct.

  Considering that this is immediately followed by " `=/oaJe>\x0A` ", and looking at the XML tags above. These are probably enclosing "name" tags.

  " `<name>syn</name>\x0A` "
- " ``8z\x05ruion:=`/versmc4>\x0A`` " Well, " `8z\x05` " and " `r` " are wrong, " `r` " and " `ion` " are correct.

  " ``:=` `` " should be wrong, " `/` " correct, " `v` " wrong, and " `ers` " correct, " `mc4` " wrong, " `>` " correct, " `\x0a` " wrong and " `</p` " correct.

  It's probably version, considering the XML pattern from above.

  So, corrected: " `<version>=</version>` " (Probably not "Version =" but rather a number, I'll deal with it later.)

```
000000d0  1c 12 a6 a7 6f 3e 0a 9a  2f 70 6c 75 67 69 6e 73  |....o>../plugins|
```

  " `</p\x1c\x12\xa6\xa7o>\x0AZ/plugins>` " This is obviously the closing "plugin" tag.
  " `</plugin>\x0A</plugins>` "

So, this looks almost totally correct, except for those version numbers. We can XOR this back against the original ciphertext, and encrypted syn.dll and
verify correctness of the EXE.

Final Pass

Review

```
00000000  3c 3f 78 6d 6c 20 76 65  72 73 69 6f 6e 3d 22 31  |<?xml version="1|
00000010  2e 30 22 3f 3e 0a 3c 62  6b 65 72 6e 65 6c 3e 0a  |.0"?>.<bkernel>.|
00000020  3c 70 6c 75 67 69 6e 73  3e 0a 3c 70 6c 75 67 69  |<plugins>.<plugi|
00000030  6e 3e 0a 3c 6e 61 6d 65  3e 64 64 6f 73 3c 2f 6e  |n>.<name>ddos</n|
00000040  61 6d 65 3e 0a 3c 76 65  72 73 69 6f 6e 3e 31 3c  |ame>.<version>1<|
00000050  2f 76 65 72 73 69 6f 6e  3e 0a 3c 2f 70 6c 75 67  |/version>.</plug|
00000060  69 6e 3e 0a 3c 70 6c 75  67 69 6e 3e 0a 3c 6e 61  |in>.<plugin>.<na|
00000070  6d 65 3e 68 74 74 70 3c  2f 6e 61 6d 65 3e 0a 3c  |me>http</name>.<|
00000080  76 65 72 73 69 6f 6e 3e  30 3c 2f 76 65 72 73 69  |version>0</versi|
00000090  6f 6e 3e 0a 3c 2f 70 6c  75 67 69 6e 3e 0a 3c 70  |on>.</plugin>.<p|
000000a0  6c 75 67 69 6e 3e 0a 3c  6e 61 6d 65 3e 73 79 6e  |lugin>.<name>syn|
000000b0  3c 2f 6e 61 6d 65 3e 0a  3c 76 65 72 73 69 6f 6e  |</name>.<version|
000000c0  3e 3d 3c 2f 76 65 72 73  69 6f 6e 3e 0a 3c 2f 70  |>=</version>.</p|
000000d0  6c 75 67 69 6e 3e 0a 3c  2f 70 6c 75 67 69 6e 73  |lugin>.</plugins|
000000e0  3e 0a 3c 63 6d 64 73 3e  0a 3c 2f 63 6d 64 73 3e  |>.<cmds>.</cmds>|
000000f0  0a 3c 70 6c 67 5f 64 61  74 61 3e 0a 3c 64 64 6f  |.<plg_data>.<ddo|
00000100  73 3e 3c 74 63 70 5f 73  69 7a 65 3e 31 30 30 30  |s><tcp_size>1000|
00000110  3c 2f 74 63 70 5f 73 69  7a 65 3e 0d 0a 3c 74 63  |</tcp_size>..<tc|
00000120  70 5f 66 72 65 71 3e 35  30 3c 2f 74 63 70 5f 66  |p_freq>50</tcp_f|
00000130  72 65 71 3e 0d 0a 3c 74  63 70 5f 74 68 72 65 61  |req>..<tcp_threa|
00000140  64 73 3e 35 3c 2f 74 63  70 5f 74 68 72 65 61 64  |ds>5</tcp_thread|
00000150  73 3e 0d 0a 3c 75 64 70  5f 73 69 7a 65 3e 31 30  |s>..<udp_size>10|
00000160  30 30 3c 2f 75 64 70 5f  73 69 7a 65 3e 0d 0a 3c  |00</udp_size>..<|
00000170  75 64 70 5f 66 72 65 71  3e 35 30 3c 2f 75 64 70  |udp_freq>50</udp|
00000180  5f 66 72 65 71 3e 0d 0a  3c 75 64 70 5f 74 68 72  |_freq>..<udp_thr|
00000190  65 61 64 73 3e 35 3c 2f  75 64 70 5f 74 68 72 65  |eads>5</udp_thre|
000001a0  61 64 73 3e 0d 0a 3c 69  63 6d 70 5f 73 69 7a 65  |ads>..<icmp_size|
000001b0  3e 31 30 30 30 3c 2f 69  63 6d 70 5f 73 69 7a 65  |>1000</icmp_size|
000001c0  3e 0d 0a 3c 69 63 6d 70  5f 66 72 65 71 3e 35 30  |>..<icmp_freq>50|
000001d0  3c 2f 69 63 6d 70 5f 66  72 65 71 3e 0d 0a 3c 69  |</icmp_freq>..<i|
000001e0  63 6d 70 5f 74 68 72 65  61 64 73 3e 35 3c 2f 69  |cmp_threads>5</i|
000001f0  63 6d 70 5f 74 68 72 65  61 64 73 3e 0d 0a 3c 68  |cmp_threads>..<h|
00000200  74 74 70 5f 66 72 65 71  3e 31 30 30 3c 2f 68 74  |ttp_freq>100</ht|
00000210  74 70 5f 66 72 65 71 3e  0d 0a 3c 68 74 74 70 5f  |tp_freq>..<http_|
00000220  74 68 72 65 61 64 73 3e  35 3c 2f 68 74 74 70 5f  |threads>5</http_|
00000230  74 68 72 65 61 64 73 3e  3c 2f 64 64 6f 73 3e 0a  |threads></ddos>.|
00000240  3c 68 74 74 70 3e 3c 68  74 74 70 5f 66 72 65 71  |<http><http_freq|
00000250  3e 33 30 3c 2f 68 74 74  70 5f 66 72 65 71 3e 0d  |>30</http_freq>.|
00000260  0a 3c 68 74 74 70 5f 74  68 72 65 61 64 73 3e 32  |.<http_threads>2|
00000270  3c 2f 68 74 74 70 5f 74  68 72 65 61 64 73 3e 3c  |</http_threads><|
00000280  2f 68 74 74 70 3e 0a 3c  73 79 6e 3e 3c 73 79 6e  |/http>.<syn><syn|
00000290  5f 66 72 65 71 3e 32 30  3c 2f 73 79 6e 5f 66 72  |_freq>20</syn_fr|
000002a0  65 71 3e 0d 0a 3c 73 79  6e 5f 74 68 72 65 61 64  |eq>..<syn_thread|
000002b0  73 3e 33 3c 2f 73 79 6e  5f 74 68 72 65 61 64 73  |s>3</syn_threads|
000002c0  3e 3c 2f 73 79 6e 3e 0a  3c 2f 70 6c 67 5f 64 61  |></syn>.</plg_da|
000002d0  74 61 3e 0a 3c 73 6c 65  65 70 66 72 65 71 3e 39  |ta>.<sleepfreq>9|
000002e0  30 30 3c 2f 73 6c 65 65  70 66 72 65 71 3e 0a 3c  |00</sleepfreq>.<|
000002f0  69 70 3e 31 32 38 2e 31  33 30 2e 35 36 2e 32 32  |ip>128.130.56.22|
00000300  3c 2f 69 70 3e 0a 3c 2f  62 6b 65 72 6e 65 6c 3e  |</ip>.</bkernel>|
00000310  0a                                                |.|
00000311
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............| ; MZ!
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 e8 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th| ; Dos stub
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  67 6f cc f5 23 0e a2 a6  22 0e a2 a6 23 0e a2 a6  |go..#..."...#...| ; oops, still off
00000090  26 02 ff a6 21 0e a2 a6  e0 01 ff a6 21 0e a2 a6  |&...!.......!...|
000000a0  30 06 ff a6 21 0e a2 a6  a0 06 ff a6 28 0e a2 a6  |0...!.......(...|
000000b0  23 0e a3 a6 05 0e a2 a6  26 02 c2 a6 24 0e a2 a6  |#.......&...$...|
000000c0  26 0e fe a6 22 0e a2 a6  26 02 f8 a6 22 0e a2 a6  |&..."...&..."...|
000000d0  52 69 63 68 23 0e a2 a6  00 00 00 00 00 00 00 00  |Rich#...........|
000000e0  00 00 00 00 00 00 00 00  50 45 00 00 4c 01 04 00  |........PE..L...| ; PE Header
000000f0  79 42 b3 48 00 00 00 00  00 00 00 00 e0 00 0e 21  |yB.H...........!|
00000100  0b 01 07 0a 00 0e 00 00  00 3a 00 00 00 00 00 00  |.........:......| ; PE Optional Header
00000110  ac 10 00 00 00 10 00 00  00 20 00 00 00 00 00 10  |......... ......|
00000120  00 10 00 00 00 02 00 00  04 00 00 00 00 00 00 00  |................|
00000130  04 00 00 00 00 00 00 00  00 70 00 00 00 04 00 00  |.........p......|
00000140  00 00 00 00 02 00 00 04  00 00 10 00 00 10 00 00  |................|
00000150  00 00 10 00 00 10 00 00  00 00 00 00 10 00 00 00  |................|
00000160  10 26 00 00 62 00 00 00  08 22 00 00 b4 00 00 00  |.&..b...."......|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  00 00 00 00 00 00 00 00  00 60 00 00 bc 00 00 00  |.........`......|
00000190  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000001c0  00 20 00 00 b4 00 00 00  00 00 00 00 00 00 00 00  |. ..............|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001e0  2e 74 65 78 74 00 00 00  00 0d 00 00 00 10 00 00  |.text...........|
000001f0  00 0e 00 00 00 04 00 00  00 00 00 00 00 00 00 00  |................|
00000200  00 00 00 00 20 00 00 60  2e 72 64 61 74 61 00 00  |.... ..`.rdata..|
00000210  72 06 00 00 00 20 00 00  00 08 00 00 00 12 00 00  |r.... ..........|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 40  |............@..@|
00000230  2e 64 61 74 61 00 00 00  70 2e 00 00 00 30 00 00  |.data...p....0..|
00000240  00 26 00 00 00 1a 00 00  00 00 00 00 00 00 00 00  |.&..............|
00000250  00 00 00 00 40 00 00 c0  2e 72 65 6c 6f 63 00 00  |....@....reloc..|
00000260  4c 01 00 00 00 60 00 00  00 02 00 00 00 40 00 00  |L....`.......@..|
00000270  00 00 00 00 00 00 00 00  00 00 00 00 40 00 00 42  |............@..B|
00000280  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000310  00                                                |.|
00000311
```

Finally...

Ok, so we now know the correct value of the mask, since we're confident that the corresponding XML is correct here.

```
000000d0  52 69 63 68 23 0e a2 a6  00 00 00 00 00 00 00 00  |Rich#...........|
000000d0  6c 75 67 69 6e 3e 0a 3c  2f 70 6c 75 67 69 6e 73  |lugin>.</plugins|
```

The correct mask to use is "23 0e a2 a6" not "22 0e a2 A6" as was our [my] first guess, but they were very close only differ by "01 00 00 00" so, the correction bitmask I used was something like:

(You could also redo everything from "Fixing the last chunk of XML" above, if you want to do all that XML correction over again.)

```
01 00 00 00 01 00 00 00  01 00 00 00 01 00 00 00
00000080  67 6f cc f5 23 0e a2 a6  22 0e a2 a6 23 0e a2 a6  |go..#..."...#...|
00000080  76 65 72 73 69 6f 6e 3e  30 3c 2f 76 65 72 73 69  |version>0</versi|
Correction:
00000080  67 6f cc f5 23 0e a2 a6  23 0e a2 a6 23 0e a2 a6  |go..#...#...#...|
00000080  76 65 72 73 69 6f 6e 3e  31 3c 2f 76 65 72 73 69  |version>1</versi|
```

So, the version number tag was still off by one, correcting to " `1` ".

For the other version number however, it's an actual unknown byte value,
dependent upon whatever was going through Visual Studio's Linker's tiny little mind at compile time.

This is the last unknown byte in the file(s). And the only time until now there is not enough information…
You could say, Well fuck it! It's probably also " `1` " like the other two version strings. (There's only a 1 in 16 chance of being wrong.)

```
          01 00 00 00 01 00 00 00  01 00 00 00 01 00 00 00
000000c0  26 0e fe a6 22 0e a2 a6  26 02 f8 a6 22 0e a2 a6  |&..."...&..."...|
000000c0  3e 3d 3c 2f 76 65 72 73  69 6f 6e 3e 0a 3c 2f 70  |>=</version>.</p|
000000c0  26 02 fe a6 22 0e a2 a6  26 02 f8 a6 22 0e a2 a6  |&..."...&..."...|
000000c0  3e 31 3c 2f 76 65 72 73  69 6f 6e 3e 0a 3c 2f 70  |>1</version>.</p|
```

You could also try against it, all known `@comp.id`'s matching the pattern " `26 ?? fe a6` ". Or, if every other byte up to the PE header is correct at this point, you can calculate the value of this byte by calculating backwards from the (known correct) hash value. (Left as an exercise for the reader.)

So, after doing all this, the finished config file that I came up with hashes to `2eddd3fa59f2bbec61415fc599e1aee8`
And so we can recover the keystream, and decrypt the first 785 bytes of the other three EXE (really DLL) files. This is what I got if you want to check your work against mine. You have been following along in your notebook this whole time, right?

```
27e95b028ddde6dbd9f58f3796b54f26 ddos.dll_first_785
d3c76705708d33f95a86da0dedbf5d9d http.dll_first_785
b4a14bdc6d19a805a9bd9008f555a2fa syn.dll_first_785
```

Anyway, if you have a large malware zoo, you can search for DLL files, which are 11264, 6657, and 16896 bytes long. And which begin with one of the three 785 byte sequences recovered above. (There's a correspondence between file length, and which 785 byte chunk to use, which should be obvious, so I won't explain.)

And if you do this, the files that you find are:

```
5991402077ab21c5e656550214298f20  ddos.dll [Live Sample]
fe9cf7b3f01816393298ff1345ca3c04  http.dll [Live Sample]
87b71080d75b5ca222fa51ce7563a615  syn.dll [Live Sample]
```

All of which I've tossed up on OC for your enjoyment education. None of them are packed, all of the imports to " `main.dll` " are referring to the original sample. Pay particular attention to how the "http.dll" looks remarkably similar to "Black Energy v1.x" .

Verification

Simple Check

Each of these three, when XOR's with its encrypted counterpart, all produce the exact same keystream. So these are the correctly decrypted files.

Check PE Headers

Are you tired of reading this yet, because I'm certainly tired of writing this. Anyway, All you need to know about PE headers:

http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx
You can sanity check them if you would like, I'm not going to bother. See Appendix F.

Check Rich Headers

The hash of the first umpteen bytes should calculate out correctly. I wrote a program to do some of these checks. See Appendix C below.

References

Absolutely everything that I could find about the "Rich" header.

http://ntcore.com/Files/richsign.htm

http://web17.webbpro.de/index.php?page=microsofts-rich-header

http://www.woodmann.com/forum/archive/index.php/t-11367.html

http://trendystephen.blogspot.com/2008/01/rich-header.html

And then I saw, http://docs.google.com/Doc?id=dcn764fg_2j54gzfc .

The Future

Newer versions of "Black Energy" — or what everyone calls "Black Energy v2" — do something completely different with their crypto, so, as I said, this trick no longer works.

Examine these two cyphertexts. These are the ones I used as examples. Notice how all but a single byte between offsets `0` and `0x80` are the same.

```
syn.dll:
00000000  81 7f 91 83 33 59 b8 82  04 4a de ba 99 87 86 0e  |....3Y...J......|
00000010  92 5b e2 78 d0 8a 59 8a  ba 4a c5 f4 59 31 48 1e  |.[.x..Y..J..Y1H.|
00000020  29 a9 86 92 2d 60 6d 76  e9 6e 84 b8 30 7a 45 3d  |)...-`mv.n..0zE=|
00000030  b6 27 30 7d 60 6e 3e 62  1a 71 59 22 ed 45 56 1e  |.'0}`n>b.qY".EV.|
00000040  24 89 84 5c 1d e6 e1 c5  c9 f1 27 22 34 c0 c8 35  |$..\......'"4..5|
00000050  fc 69 10 02 e4 31 4d 88  fc f4 38 8b 38 04 f4 00  |.i...1M...8.8...|
00000060  cb de 98 7c c8 fc f8 1b  2e 32 7a 9a 66 d6 16 41  |...|.....2z.f..A|
00000070  20 89 81 2d 04 68 f7 b0  7e da 60 30 62 0e 5d e2  | ..-.h..~.`0b.].|
00000080  df 6e 86 a8 39 1a eb 0b  c2 6d cc ab b2 2c 46 3a  |.n..9....m...,F:|
00000090  6f 46 01 8c b0 f7 f2 d1  06 74 be 80 68 a0 bd 0c  |oF.......t..h...|
http.dll:
00000000  81 7f 91 83 33 59 b8 82  04 4a de ba 99 87 86 0e  |....3Y...J......|
00000010  92 5b e2 78 d0 8a 59 8a  ba 4a c5 f4 59 31 48 1e  |.[.x..Y..J..Y1H.|
00000020  29 a9 86 92 2d 60 6d 76  e9 6e 84 b8 30 7a 45 3d  |)...-`mv.n..0zE=|
00000030  b6 27 30 7d 60 6e 3e 62  1a 71 59 22 fd 45 56 1e  |.'0}`n>b.qY".EV.|
00000040  24 89 84 5c 1d e6 e1 c5  c9 f1 27 22 34 c0 c8 35  |$..\......'"4..5|
00000050  fc 69 10 02 e4 31 4d 88  fc f4 38 8b 38 04 f4 00  |.i...1M...8.8...|
00000060  cb de 98 7c c8 fc f8 1b  2e 32 7a 9a 66 d6 16 41  |...|.....2z.f..A|
00000070  20 89 81 2d 04 68 f7 b0  7e da 60 30 62 0e 5d e2  | ..-.h..~.`0b.].|
00000080  55 bf 8d c9 b3 cb e0 6a  48 bc c7 ca 38 fd 4d 5b  |U......jH...8.M[|
00000090  e5 97 58 ed 3b 26 f9 b0  4a a6 b7 e1 e1 71 b6 6d  |..X.;&..J....q.m|
```

Now look at these three newer cyphertexts, generated by newer versions of the bot. There is nothing in common.

```
EXE 1:
00000000  e9 e1 c4 48 3b 42 c2 94  df 90 50 ef d2 8f 10 78  |...H;B....P....x|
00000010  8f 5c f4 c8 d0 0d a4 5c  7c 14 3a 19 f9 d3 f9 7c  |.\.....\|.:....||
00000020  85 48 6c fa 1d 44 c3 cf  37 8c 19 e3 17 1c d8 b6  |.Hl..D..7.......|
00000030  c8 98 34 70 e6 57 3c 38  0e 9e fa 9e 27 a2 22 4b  |..4p.W<8....'."K|
00000040  9e 7b 7e d3 5b a4 2b 71  b2 1e 5d fc 69 ee 0e be  |.{~.[.+q..].i...|
00000050  ab 20 68 b3 f7 d7 2b 3f  6c 9d 90 0b 62 85 46 e2  |. h...+?l...b.F.|
00000060  00 d9 b2 8e e8 90 0c 06  74 af 2a 70 de c9 ee d7  |........t.*p....|
00000070  7d f3 59 16 2b fe 86 7d  2c 39 b8 68 59 dc 79 c2  |}.Y.+..},9.hY.y.|
00000080  83 99 3c 7d fd 1d f2 ad  3b 9e 2a 18 14 b0 15 95  |..<}....;.*.....|
00000090  ee 3f 3a ab 04 85 be 9d  e5 27 c1 02 aa 26 aa 55  |.?:......'...&.U|
EXE 2:
00000000  4f 57 70 aa 8c 86 a0 54  b5 d6 14 d0 3e d5 e1 40  |OWp....T....>..@|
00000010  12 20 c1 44 02 61 26 e1  1b 0a 4d 3a 7b 2a 6a f1  |. .D.a&...M:{*j.|
00000020  02 7c 6f 29 d1 5b c6 11  10 ee d1 c6 8a 03 11 c3  |.|o).[..........|
00000030  42 e1 21 3c 62 2d 98 e6  0d 9c 40 91 48 34 8f b0  |B.!<b-....@.H4..|
00000040  4a 32 70 a3 20 34 e7 02  67 19 eb 2a 0c b5 ed ec  |J2p. 4..g..*....|
00000050  83 3a 76 1e 49 b3 13 34  02 82 2f e5 6c 2e be 74  |.:v.I..4../.l..t|
00000060  13 03 4d 07 6f 33 16 68  11 c6 a9 02 c4 3f 77 e2  |..M.o3.h.....?w.|
00000070  5d 34 0a 1e ef cd 2f 50  d4 76 e2 58 a0 c7 af 6d  |]4..../P.v.X...m|
00000080  3d 74 a0 99 ef 75 e4 5e  07 d2 0f 96 a9 06 e7 96  |=t...u.^........|
00000090  ce 62 7d 89 4b 1f 08 6d  e3 f5 8a fb dc 92 83 87  |.b}.K..m........|
EXE 3:
00000000  1f 60 5a f2 3e c5 25 e1  62 52 9a 2b f9 1d 4f d4  |.`Z.>.%.bR.+..O.|
00000010  77 84 47 d4 8b a4 d9 ea  a8 da a8 12 ec d9 ea 6f  |w.G............o|
00000020  67 ac 85 27 8f 25 5f 7f  cf f9 19 ec 45 0f d3 c5  |g..'.%_.....E...|
00000030  ef 19 cf 52 0c 5d 94 c8  48 8c 34 a7 93 c5 45 a7  |...R.]..H.4...E.|
00000040  74 55 fb fa 12 52 47 ca  87 62 49 62 b2 bc 18 0a  |tU...RG..bIb....|
00000050  4e 93 cb d1 34 2b f3 4d  86 9f 2a 1e 13 4f 0a c3  |N...4+.M..*..O..|
00000060  93 26 e8 49 32 40 e1 22  f4 16 69 fd a0 a7 b7 ee  |.&.I2@."..i.....|
00000070  1e 83 a5 4f fd ff c9 59  cb 32 b9 8e a1 8c 73 8f  |...O...Y.2....s.|
00000080  e0 84 49 50 d9 56 79 16  1c 5b 27 0d 95 5a a8 4d  |..IP.Vy..['..Z.M|
00000090  c5 2f 73 8e 7a 72 19 b8  74 94 22 d1 45 fb 9c 4c  |./s.zr..t.".E..L|
```

The End

Yay! I'm done.

Appendix A

When I say `file ^ file = file`, that means

each pair of bytes, from the same offset in both files, together, and write the result to the same offset in the third. Formally, think of it as something like

$$\sum_{i=0}^{n} output_i = file_i^1 \oplus file_i^2$$

You could also say the same thing about summing all the bits over `GF(2)`. But pretend that I didn't just say all that.

So, File "`x`" is a stream of bytes, File "`y`" is a stream of bytes, `n = min(len(x),len(y))`

Vaguely almost like this:

```
while ((0!=sysread(IN1, $inbuf1, 1))&&(0!=sysread(IN2, $inbuf2, 1))) {
$outbuf = $inbuf1 ^ $inbuf2;
syswrite(OUT, $outbuf, 1);
}
```

Appendix B

This is how I XOR files together. You might find it useful.

```
use Fcntl;
my $infile1 = shift;
my $infile2 = shift;
my $outfile = shift;
my $inbuf1;
my $inbuf2;
my $outbuf;
print("DEBUG: $infile1 ^ $infile2 -> $outfile\n");
sysopen(IN1, $infile1, O_RDONLY);
sysopen(IN2, $infile2, O_RDONLY);
sysopen(OUT, $outfile, O_WRONLY|O_CREAT);
binmode(IN1);
binmode(IN2);
binmode(OUT);
while ((0!=sysread(IN1, $inbuf1, 1))&&(0!=sysread(IN2, $inbuf2, 1))) {
$outbuf = $inbuf1 ^ $inbuf2;
syswrite(OUT, $outbuf, 1);
}
close($infile1);
close($infile2);
close($outfile);
```

Appendix C

If you need to un-HTML-ize this, the only encoding I did on it was:

`sed 's/&/\&amp;/g' hash.pl | sed 's/</\&lt;/g' | sed 's/>/\&gt;/g'`

```
#!/usr/bin/perl -w
# Julia Wolf
# Rich header hash checker.
# Public Domain
# Version 0.0.1 alpha
# Not really tested, use at own risk and all that jazz.
# Usage:
# perl this.pl some.exe
use strict;
my $file = shift or die;
sub rol {
my $number = shift;
my $bitshift = (shift) % 32;
return ( 0x0FFFFFFFF&($number <<       $bitshift  )        ) |
( 0x0FFFFFFFF&($number >> (32 - $bitshift) )        ) ;
}
# unused
sub ror {
my $number = shift;
my $bitshift = (shift) % 32;
return ($number >> $bitshift) | ($number << (32 - $bitshift) );
}
my $start = 0x80; # TODO
my $data = `cat $file`; # Warning: Not portable code
my @bytes = split(//,$data);
map {$_ = unpack("C",$_); } @bytes;
my @dwords = unpack("V*", $data);
#   .text:004651D6 xormaskloop1 :                 ; CODE XREF: IMAGE::CbBuildProdidBlock(void * *)+105j
#   .text:004651D6       movzx edi, byte ptr [ebx+eax] ; edi = (BYTE) PointerToPE[eax]
#   .text:004651DA       mov cl, al              ; low byte of loop counter in cl
#   .text:004651DC       rol edi, cl             ; rotates left the current byte of PointerToPE
#   .text:004651DC                               ; with the low byte of the loop counter
#   .text:004651DE       add eax, 1              ; increment eax
#   .text:004651E1       add esi, edi            ; adds the result of the rol to the xor mask
#   .text:004651E3       cmp eax, edx            ; is counter < initial xor mask value?
#   .text:004651E5       jb short xormaskloop1   ; if so, goes on with the loop
#   .text:004651E7
my $mask = $start;
for (my $i=0; $i<$start; $i++) { next if ( $i>=0x3C && $i<=0x3F );
last if ($i>=$#bytes);
$mask = $mask + rol($bytes[$i],$i);
$mask = $mask & 0xFFFFFFFF;
}
printf ("%016x mask\n", $mask);
#00000080  44 61 6e 53  00 00 00 00  00 00 00 00  00 00 00 00  |DanS............|
#00000090  05 0c 5d 00  02 00 00 00  c3 0f 5d 00  02 00 00 00  |..].......].....|
#000000a0  13 08 5d 00  02 00 00 00  83 08 5d 00  0b 00 00 00  |..].......].....|
#000000b0  00 00 01 00  26 00 00 00  05 0c 60 00  07 00 00 00  |....&.....`.....|
#000000c0  05 0c 5c 00  01 00 00 00  05 0c 5a 00  01 00 00 00  |...\......Z.....|
#000000d0  71 67 c1 ce  00 00 00 00  23 0e a2 a6  23 0e a2 a6  |qg......#...#...|
#00000080  67 6f cc f5  23 0e a2 a6  23 0e a2 a6  23 0e a2 a6  |go..#...#...#...|
#00000090  26 02 ff a6  21 0e a2 a6  e0 01 ff a6  21 0e a2 a6  |&...!.......!...|
#000000a0  30 06 ff a6  21 0e a2 a6  a0 06 ff a6  28 0e a2 a6  |0...!.......(...|
#000000b0  23 0e a3 a6  05 0e a2 a6  26 02 c2 a6  24 0e a2 a6  |#.......&...$...|
#000000c0  26 02 fe a6  22 0e a2 a6  26 02 f8 a6  22 0e a2 a6  |&..."...&..."...|
#000000d0  52 69 63 68  23 0e a2 a6  00 00 00 00  00 00 00 00  |Rich#...........|
my $thingy = $mask;
#   .text:004651F0 xormaskloop2 :                 ; CODE XREF: IMAGE::CbBuildProdidBlock(void * *)+11Ej
#   .text:004651F0       mov edx, [eax+ 4 ]        ; edx = data1
#   .text:004651F3       mov cl, [eax+ 8 ]         ; cl = (BYTE) data2
#   .text:004651F6       mov eax, [eax]          ; eax = next list item
#   .text:004651F8       rol edx, cl             ; rotates left edx with cl
#   .text:004651FA       add esi, edx            ; adds the result of the rol to the xor mask
#   .text:004651FC       test eax, eax           ; pointer = 0?
#   .text:004651FE       jnz short xormaskloop2  ; if not, goes on with the loop
my $pe_offset = $dwords[15];
if ( $pe_offset ) {
if ( 0x00004550 == $dwords[$pe_offset/4] ) {
print ("PE OK\n");
} else {
print ("PE bad\n");
}
} else {
print ("PE missing\n");
}
my $rich_offset = 0;   #bytes
do {
$rich_offset++; # in dwords
} while ( ($rich_offset<$#dwords)&&(0x68636952 != $dwords[$rich_offset]) ); # undef check if past end
printf ("%016x rich off\n", $rich_offset*4);
my $checksum = $dwords[$rich_offset+1];
if ( $checksum ) {
printf ("%016x checksum\n", $checksum);
} else {
print ("checksum missing\n");
}
my $ugh=0;
for (my $i=0 ; $i<=$rich_offset; $i++) {
```

```
if ( $checksum ) {
if (0x536e6144 == ($dwords[$i] ^ $checksum)) {
print ("DanS OK $i\n");
$ugh = $i;
last;
}
} else {
print ("No DanS, checksum missing\n");
}
}
my $count = 0;
for (my $i=$ugh+4; $i<=$rich_offset-1; $i+=2) {
print("Ignore these version numbers for now. The top WORD means something completely different.\n"); #TODO
# printf("comp.id\t%i.0.%i ? %i x %i\n",(($dwords[$i]^$checksum)&0xFFFF),((($dwords[$i]^$checksum)>>16)&0x000F),
((($dwords[$i]^$checksum)>>17)&0x000F), ($dwords[$i+1]^$checksum) );
# printf("comp.id\t%i.00.%i (%i?) x %i\n",((($dwords[$i]^$checksum)>>16)&0x000F),(($dwords[$i]^$checksum)&0xFFFF),
((($dwords[$i]^$checksum)>>17)&0x000F), ($dwords[$i+1]^$checksum) );
# printf("comp.id\t%i.%i.%i (%i?) x %i\n",((($dwords[$i]^$checksum)>>16)&0x000F),((($dwords[$i]^$checksum)>>24)&0x00FF),
(($dwords[$i]^$checksum)&0xFFFF),((($dwords[$i]^$checksum)>>17)&0x000F), ($dwords[$i+1]^$checksum) );
printf("comp.id\t%08x\t%i.%i.%i Occurs: %i\n",
($dwords[$i]^$checksum),
((($dwords[$i]^$checksum)&0x000F0000)>>16), # 13 major
((($dwords[$i]^$checksum)&0x00F00000)>>19),   # 10 minor
(($dwords[$i]^$checksum)&0x0000FFFF),      # 3077 build
#        ((($dwords[$i]^$checksum)>>20)&0x000F), #
($dwords[$i+1]^$checksum) );            # 2 occurence
# This doesn't help...
#   //
#   // Symbol format.
#   //
#
#   typedef struct _IMAGE_SYMBOL {
#       union {
#           BYTE    ShortName[8];
#           struct {
#               DWORD   Short;     // if 0, use LongName
#               DWORD   Long;      // offset into string table
#           } Name;
#           DWORD   LongName[2];    // PBYTE [2]
#       } N;
#       DWORD   Value;
#       SHORT   SectionNumber;
#       WORD    Type;
#       BYTE    StorageClass;
#       BYTE    NumberOfAuxSymbols;
#   } IMAGE_SYMBOL;
#   typedef IMAGE_SYMBOL UNALIGNED *PIMAGE_SYMBOL;
#
#   #define IMAGE_SIZEOF_SYMBOL                  18
#   //
#   // Section values.
#   //
#   // Symbols have a section number of the section in which they are
#   // defined. Otherwise, section numbers have the following meanings:
#   //
#
#   #define IMAGE_SYM_UNDEFINED           (SHORT)0          // Symbol is undefined or is common.
#   #define IMAGE_SYM_ABSOLUTE            (SHORT)-1         // Symbol is an absolute value.
#   #define IMAGE_SYM_DEBUG               (SHORT)-2         // Symbol is a special debug item.
#   #define IMAGE_SYM_SECTION_MAX         0xFEFF            // Values 0xFF00-0xFFFF are special
#
$thingy = $thingy + rol( $dwords[$i] ^ $checksum, $dwords[$i+1] ^ $checksum );
$thingy = $thingy & 0xFFFFFFFF;
$count++; # item count
}
printf ("%016x thingy\n", $thingy);
printf ("%016x diff\n", $checksum - $thingy ) if ( $checksum );
printf ("%016x xor\n", $checksum ^ $thingy ) if ( $checksum );
my $list = ( $count ) * 8;
printf ("%016x list len bytes\n", $list );
printf ("%016x DanS start bytes\n", $ugh*4 );
printf ("%016x list start bytes\n", $ugh*4 + 0x10);
printf ("%016x list end bytes\n", $list+($ugh*4) + 0x10 );
print ("PE == end") if () ;
my $padding = ( (  (($thingy >> 5) % 3)  + $count ) * 8) + 0x20;
printf ("%016x pad\n", $padding );
printf ("%016x end\n", ($ugh*4)+$padding);
print ("PE == end") if ($pe_offset)&&( (($ugh*4)+$padding) == ($pe_offset) ) ;
print "\n";
exit(0);
```

Appendix D

```
2eddd3fa59f2bbec61415fc599e1aee8  1253064020.308867 88.214.243.45 80 192.168.0.2 1038 785 Decrypted Config File
5991402077ab21c5e656550214298f20  1253064023.563605 88.214.243.45 80 192.168.0.2 1039 11264 Decrypted "ddos" EXE
fe9cf7b3f01816393298ff1345ca3c04  1253064029.663532 88.214.243.45 80 192.168.0.2 1040 6657 Decrypted "http" EXE
87b71080d75b5ca222fa51ce7563a615  1253064034.512609 88.214.243.45 80 192.168.0.2 1041 16896 Decrypted "syn" EXE
adae2ddc6ec2cedf9d575b48267b53a4  6657  ddos_x_http
8496ee21928543ee9b49b8df5e1c861b  11264 ddos_x_syn
04c978be26ee36f7cc050795dd2c648b  6657  http_x_syn
c61e182c9b67ab9067138deac9f831a5  785   xml_x_ddos
ddb168742d95068046340ee18fa50dbc  785   xml_x_http
0880df84c5b886cbdd0e0be01deed2f6  785   xml_x_syn
```

Appendix E

Look for the value of the `@comp.id` symbol, in the output from either of these:

```
nm coff.obj
```

```
dumpbin /SYMBOLS coff.obj
```

Build	`@comp.id`	Notes (probably wrong)
2179	`000F0883`	Version 7.10.2179 (DDK 3790?)
3052	`005F0BEC`	Version 7.10.3052 ?
3077	`000F0C05`	Version 7.10.3077 (Visual Studio 2003)
3077	`005D0C05`	Version 7.10.3077
3077	`005F0C05`	Version 7.10.3077
3077	`00600C05`	Version 7.10.3077
4035		7.10.4035 (DDK 3790.1830)
8155	`000B1FDB`	Version 6.00.8155 ?
8168	`00041FE8`	Version 6.00.8168 ?
8168	`000A1FE8`	
8168	`000B1FE8`	Version 6.00.8168 ?
8444	`001220FC`	Version 6.14.8444
8447	`000420ff`	Version 6.00.8447 ?
8966	`000A2306`	
9044	`00312354`	
9466	`004024FA`	Version 7.00.9466 ?
21022	`0083521E`	VS2008 9.00.21022.08 ?
21022	`0093521E`	9.00.21022 ?
30729	`00837809`	VS2008 (9.0) SP1 15.00.30729.01 ?
30729	`00937809`	VS2008 (9.0) SP1 15.00.30729.01 ?
50727	`006DC627`	VS2005 for C objects; 8.00.50727.762 VS2005 (8.0) SP1 14.00.50727.762
50727	`006EC627`	VS2005 (via 8.00.50727.762)

50727	`006EC627`	VS2005 for c++ objects; 8.00.50727.762? or 8.00.50727.42?
50727	`007BC627`	VS2005 (8.0)

Appendix F

```
syn.dll:     file format efi-app-ia32
syn.dll
architecture: i386, flags 0x0000010b:
HAS_RELOC, EXEC_P, HAS_DEBUG, D_PAGED
start address 0x100010ac
Characteristics 0x210e
executable
line numbers stripped
symbols stripped
32 bit words
DLL
Time/Date               Mon Aug 25 16:38:33 2008
Magic                   010b    (PE32)
MajorLinkerVersion      7
MinorLinkerVersion      10
SizeOfCode              00000e00
SizeOfInitializedData   00003a00
SizeOfUninitializedData 00000000
AddressOfEntryPoint     00000000000010ac
BaseOfCode              0000000000001000
BaseOfData              0000000000002000
ImageBase               0000000010000000
SectionAlignment        0000000000001000
FileAlignment           0000000000000200
MajorOSystemVersion     4
MinorOSystemVersion     0
MajorImageVersion       0
MinorImageVersion       0
MajorSubsystemVersion   4
MinorSubsystemVersion   0
Win32Version            00000000
SizeOfImage             00007000
SizeOfHeaders           00000400
CheckSum                00000000
Subsystem               00000002        (Windows GUI)
DllCharacteristics      00000400
SizeOfStackReserve      0000000000100000
SizeOfStackCommit       0000000000001000
SizeOfHeapReserve       0000000000100000
SizeOfHeapCommit        0000000000001000
LoaderFlags             00000000
NumberOfRvaAndSizes     00000010
The Data Directory
Entry 0 0000000000002610 00000062 Export Directory [.edata (or where ever we found it)]
Entry 1 0000000000002208 000000b4 Import Directory [parts of .idata]
Entry 2 0000000000000000 00000000 Resource Directory [.rsrc]
Entry 3 0000000000000000 00000000 Exception Directory [.pdata]
Entry 4 0000000000000000 00000000 Security Directory
Entry 5 0000000000006000 000000bc Base Relocation Directory [.reloc]
Entry 6 0000000000000000 00000000 Debug Directory
Entry 7 0000000000000000 00000000 Description Directory
Entry 8 0000000000000000 00000000 Special Directory
Entry 9 0000000000000000 00000000 Thread Storage Directory [.tls]
Entry a 0000000000000000 00000000 Load Configuration Directory
Entry b 0000000000000000 00000000 Bound Import Directory
Entry c 0000000000002000 000000b4 Import Address Table Directory
Entry d 0000000000000000 00000000 Delay Import Directory
Entry e 0000000000000000 00000000 CLR Runtime Header
Entry f 0000000000000000 00000000 Reserved
There is an import table in .rdata at 0x10002208
The Import Tables (interpreted .rdata section contents)
vma:            Hint    Time      Forward  DLL       First
Table   Stamp     Chain    Name      Thunk
There is an export table in .rdata at 0x10002610
PE File Base Relocations (interpreted .reloc section contents)
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
0 .text         00000d00  10001000  10001000  00000400  2**2
CONTENTS, ALLOC, LOAD, READONLY, CODE
1 .rdata        00000672  10002000  10002000  00001200  2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA
2 .data         00002600  10003000  10003000  00001a00  2**2
CONTENTS, ALLOC, LOAD, DATA
3 .reloc        0000014c  10006000  10006000  00004000  2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA
SYMBOL TABLE:
no symbols
```

Julia Wolf @ FireEye Malware Intelligence Lab

Questions/Comments to `research` [@] `fireeye` [.] `com`