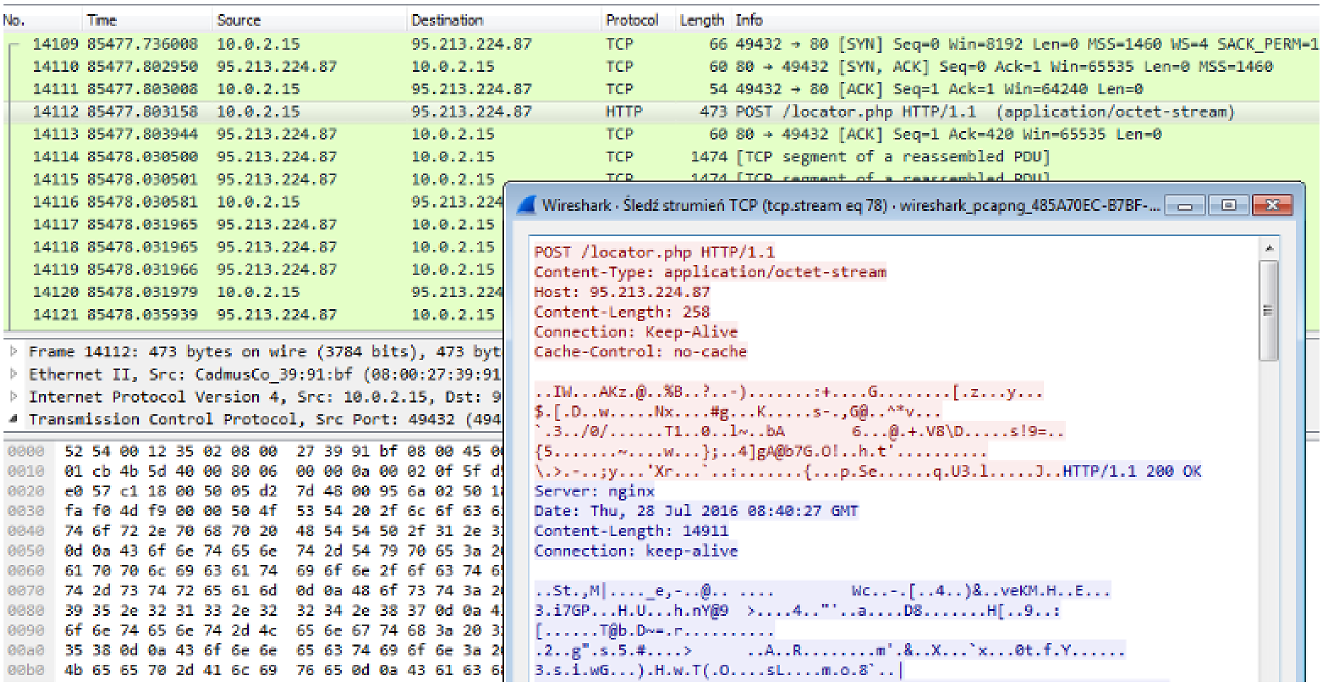
Necurs – hybrid spam botnet

cert.pl/en/news/single/necurs-hybrid-spam-botnet/

Necurs is one of the biggest botnets in the world – according to MalwareTech there are a couple millions of infected computers, several hundred thousand of which are online at any given time. Compromised computers send spam email to large number of recipients – usually the messages are created to look like a request to check invoice details or to confirm purchase. The attachments contain packed scripts which install malware when ran. Currently, the dropped ransomware is Locky, which encrypts the hard disk and then asks for money (often in Bitcoin) in order to retrieve the original files. Necurs is an example of hybrid network in terms of Command and Control architecture – a mixture of centralized model (which allows to quickly control the botnet), with peer-to-peer (P2P) model, making it next to impossible to take over the whole botnet by shutting down just a single server. For those reasons, the huge success of Necurs is no surprise.

Behaviour

The malware attempts to connect to the C2 server, whose IP address is retrieved in a number of different ways. First, a couple of domains or raw IP addresses are embedded in the program resources (in encrypted form – more about this in the technical analysis section). If the connection fails, Necurs runs domain generation algorithm, crafting up to 2048 pseudorandom names, generation of which depends on current date and seed hardcoded in encrypted resources, and tries them all in a couple of threads. If any of them resolves and responds using the correct protocol, it is saved as a server address. Otherwise, if all these methods fail, C2 domain is retrieved from the P2P network – the initial list of about 2000 peers (in form of IP+port pairs) is hardcoded in the binary. During analysis, Necurs used the last method, since none of the DGA domains was responding. It is however possible, that in the future the botnet's author will start to register these domains – a new list of potential addresses is generated every 4 days.



```
No.   Time         Source          Destination     Protocol  Length Info
14109 85477.736008 10.0.2.15       95.213.224.87   TCP         66 49432 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
14110 85477.802950 95.213.224.87   10.0.2.15       TCP         60 80 → 49432 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
14111 85477.803008 10.0.2.15       95.213.224.87   TCP         54 49432 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
14112 85477.803158 10.0.2.15       95.213.224.87   HTTP       473 POST /locator.php HTTP/1.1  (application/octet-stream)
14113 85477.803944 95.213.224.87   10.0.2.15       TCP         60 80 → 49432 [ACK] Seq=1 Ack=420 Win=65535 Len=0
14114 85478.030500 95.213.224.87   10.0.2.15       TCP       1474 [TCP segment of a reassembled PDU]
14115 85478.030501 95.213.224.87   10.0.2.15       TCP       1474 [TCP segment of a reassembled PDU]
14116 85478.030581 10.0.2.15       95.213.224
14117 85478.031965 95.213.224.87   10.0.2.15
14118 85478.031965 95.213.224.87   10.0.2.15
14119 85478.031966 95.213.224.87   10.0.2.15
14120 85478.031979 10.0.2.15       95.213.224
14121 85478.035939 95.213.224.87   10.0.2.15
```

```
▷ Frame 14112: 473 bytes on wire (3784 bits), 473 byt
▷ Ethernet II, Src: CadmusCo_39:91:bf (08:00:27:39:91
▷ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 9
◢ Transmission Control Protocol, Src Port: 49432 (494
```

```
0000  52 54 00 12 35 02 08 00  27 39 91 bf 08 00 45 0
0010  01 cb 4b 5d 40 00 80 06  00 00 0a 00 02 0f 5f d
0020  e0 57 c1 18 00 50 05 d2  7d 48 00 95 6a 02 50 1
0030  fa f0 4d f9 00 00 50 4f  53 54 20 2f 6c 6f 63 6
0040  74 6f 72 2e 70 68 70 20  48 54 54 50 2f 31 2e 3
0050  0d 0a 43 6f 6e 74 65 6e  74 2d 54 79 70 65 3a 2
0060  61 70 70 6c 69 63 61 74  69 6f 6e 2f 6f 63 74 6
0070  74 2d 73 74 72 65 61 6d  0d 0a 48 6f 73 74 3a 2
0080  39 35 2e 32 31 33 2e 32  32 34 2e 38 37 0d 0a 4
0090  6f 6e 74 65 6e 74 2d 4c  65 6e 67 74 68 3a 20 3
00a0  35 38 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 2
00b0  4b 65 65 70 2d 41 6c 69  76 65 0d 0a 43 61 63 6
```

Wireshark · Śledź strumień TCP (tcp.stream eq 78) · wireshark_pcapng_485A70EC-B7BF-...

```
POST /locator.php HTTP/1.1
Content-Type: application/octet-stream
Host: 95.213.224.87
Content-Length: 258
Connection: Keep-Alive
Cache-Control: no-cache

..IW...AKz.@..%B..?..-)........:+....G.........[.z...y...
$.[.D..w.....Nx....#g...K.....s-.,G@..^*v...
`.3../0/......T1..0..l~..bA       6...@.+.V8\D.....s!9=..
{5........~....w...};..4]gA@b7G.O!..h.t'..........
\.>.-..;y...'Xr...`..:........{...p.Se......q.U3.l.....J..HTTP/1.1 200 OK
Server: nginx
Date: Thu, 28 Jul 2016 08:40:27 GMT
Content-Length: 14911
Connection: keep-alive

..St.,M|....._e,-..@.. ....        Wc..-.[..4..)&..veKM.H..E...
3.i7GP...H.U...h.nY@9  >....4.."'..a....D8.......H[..9..:
[......T@b.D~=.r..........
.2..g".s.5.#....>      ..A..R........m'.&..X...`x...0t.f.Y......
3.s.i.wG...).H.w.T(.O....sL....m.o.8`..|
```

After establishing a successful connection to the C2, Necurs downloads (using custom protocol over HTTP) a list of information – from now on, I will call them "resources". Every resource is identified by a constant 64-bit number. It is quite likely a hash of some sensible name used in source code, but after compilation we obviously cannot access it. Nonetheless, analyzing how these resources are used in the code, we could map some of the IDs to useful names.

Examples of information sent by the C2 include: new P2P neighborhood (ca. 2000 IP:port pairs), new C2 domain list, sleep command (usually about twenty minutes or so), or request to download and run DLL module. Every request we receive contains the sleep request – it is probably a way to reduce the server's load.

The analyzed binary did not contain what we really sought – mail sending routine. As it turns out, that functionality is in one of the dropped DLLs. Unfortunately, it was written in C++, which increased code size (because the author used C++ templates) and therefore, slowed down reverse engineering. For that reason, we mostly used behavioral analysis – debugging malware and observing sent data (before the encryption of course).

As we could see, the payload is formatted as JSON. Unfortunately all of the keys were obfuscated and it was impossible to discover their meaning just by the name – for example "dg3XGB9" corresponds to the current Unix time. There are a couple of message formats, but not all of them are really interesting. The most important was the request for mail to be sent and of course the server's response. The text Necurs sends is not just literal email – a simple script language is used to randomize them:

We can see, that the script supports local variables (declared by *%%var* directive), predefined functions, such as *randnum*, but there are also references to external data – e.g. *[file.doc]* – these variables are downloaded in a separate request. We checked the attachments, and despite the name (*file.doc*), these are ZIP archives containing a single JS file. When executed, they download and run Zepto, a rather new variant of Locky ransomware.

Technical analysis

Necurs uses a couple of anti-analysis techniques. For example, every C2 connection is attempted randomly: either to the address given in function argument, or to the address being a hash of the argument. Virtualization is detected using instructions such as "*vmcpuid"*, or "*in al"*. Some malware analysis environments can also fail on checking whether Facebook and random domain resolve to the same IP address. Many texts and binary resources are encrypted – communication with peers and C2 as well.

Resources

Constants in the binary are hidden in a separate section – the file contains two named "*.reloc*" sections, the second of which contains resources. First four bytes of that section are interpreted as a decryption key, and the resources themselves start at offset 0x18. Every byte is xored with key, which changes according to LCG algorithm: *K'=K*0x19661f+0x3c6ef387*. After decryption, the data is a list of concatenated structures of the following format:

Last field size is *size>>8*. Every resource has its unique identifier – examples of resources are initial peer or C2 communication keys or initial peer neighborhood list.

P2P communication

P2P communication is unfortunately much more complicated. All information exchange happens over UDP protocol. The outermost layer of communication is:

Wrapped data are encrypted using the key calculated as a sum of the *key* field and the first 32 bits of the public key contained in file resources. The homemade encryption algorithm is equivalent to the following Python code:

Checksum sent as second field in the structure is simply a final value of key after encryption. The decrypted data have the following form:

Size of *data* is *size_flag>>4*, and the type of the message is determined by four least significant bits of that field. For example, first message ("greeting") has these bits all zeroed.

The next stage depends on the message type. For the first message, the structure is:

Should the peer respond, the message has the following form:

The whole message is signed using key from file resources. The most important field of this structure is *resources* – list of resources in the same format as described in “Resources” section. Interestingly, peers don’t send new neighborhood list – these are sent by the C2 itself. The most likely reason for this measure is avoiding P2P poisoning, since it is known that peer list received from the main server is authorized and correct.

C2 communication

C2 protocol is vaguely similar to the P2P one, but encryption routines and structures it uses are a bit different – also, the underlying protocol is HTTP (POST payload) instead of raw UDP sockets. The first stage is exactly the same (*outer_layer* structure), with different constants in encryption algorithm:

Decrypted data is of the following structure:

The first field contains randomly generated 8 bytes, probably to increase entropy of sent data and to make it harder to see patterns like common initial bytes across messages.

Contents of the *payload* field (perhaps compressed, depending on the second bit of *flags*) depends on message type (*command* field). If it is file download request (*command*=1), the payload is simply the SHA-1 hash of the requested file. On the other hand, if the whole message is a periodic command request (*command*=0), the payload structure is much more complex – again, a kind of list of resources, but with different structure. Every resource has the following general form (can be thought of as header):

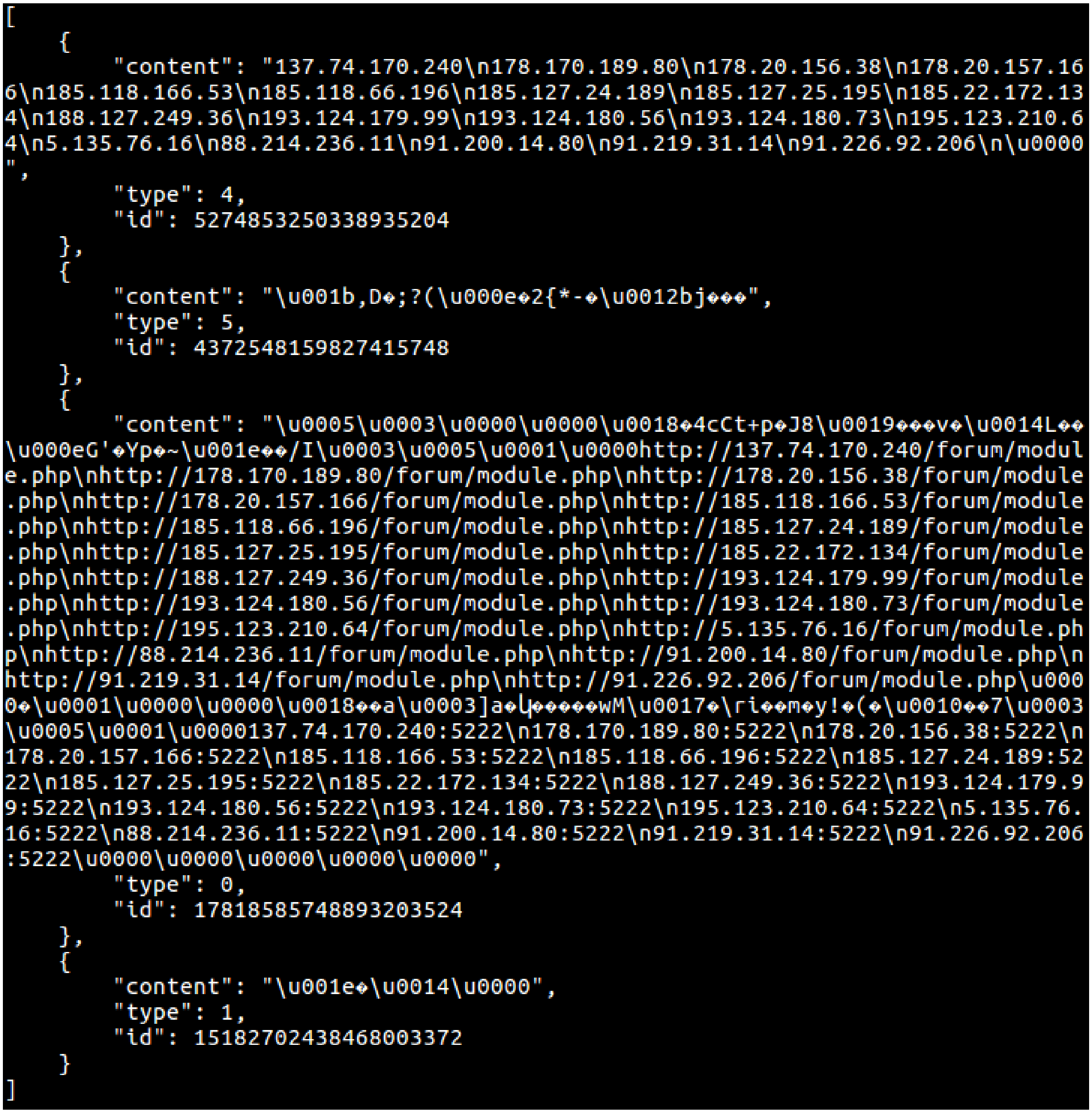
Depending on resource *type*, data has different format:

Type 4 is usually used to send text data, which is probably the reason of the resource size being increased by one (for null terminator). Client sends a list of such resources to the C2. We were able to identify the meaning of some of them:

- DGA seed
- number of seconds since malware start
- Unix timestamp of malware start
- OS version and its default language
- computer’s IP (local if behind NAT)
- UDP port used to listen for P2P connections
- custom hash of current peer list

The server response uses a very similar format. The payload also depends on request type – if it was 1 (download file request), server responds with that file’s contents (usually compressed, depending on flags). For command request, the server response is the list of resources in the same format as above. Some of these resources can be interpreted as

commands to be executed, for example “sleep N milliseconds” or “log off the user” (although I did not see the latter used in the wild).
Sample (parsed) resource list received from C2:



```
[
    {
        "content": "137.74.170.240\n178.170.189.80\n178.20.156.38\n178.20.157.16
6\n185.118.166.53\n185.118.66.196\n185.127.24.189\n185.127.25.195\n185.22.172.13
4\n188.127.249.36\n193.124.179.99\n193.124.180.56\n193.124.180.73\n195.123.210.6
4\n5.135.76.16\n88.214.236.11\n91.200.14.80\n91.219.31.14\n91.226.92.206\n\u0000
",
        "type": 4,
        "id": 5274853250338935204
    },
    {
        "content": "\u001b,D�;?(\u000e�2{*-�\u0012bj���",
        "type": 5,
        "id": 4372548159827415748
    },
    {
        "content": "\u0005\u0003\u0000\u0000\u0018�4cCt+p�J8\u0019���v�\u0014L��
\u000eG'�Yp�~\u001e��/I\u0003\u0005\u0001\u0000http://137.74.170.240/forum/modul
e.php\nhttp://178.170.189.80/forum/module.php\nhttp://178.20.156.38/forum/module
.php\nhttp://178.20.157.166/forum/module.php\nhttp://185.118.166.53/forum/module
.php\nhttp://185.118.66.196/forum/module.php\nhttp://185.127.24.189/forum/module
.php\nhttp://185.127.25.195/forum/module.php\nhttp://185.22.172.134/forum/module
.php\nhttp://188.127.249.36/forum/module.php\nhttp://193.124.179.99/forum/module
.php\nhttp://193.124.180.56/forum/module.php\nhttp://193.124.180.73/forum/module
.php\nhttp://195.123.210.64/forum/module.php\nhttp://5.135.76.16/forum/module.ph
p\nhttp://88.214.236.11/forum/module.php\nhttp://91.200.14.80/forum/module.php\n
http://91.219.31.14/forum/module.php\nhttp://91.226.92.206/forum/module.php\u000
0�\u0001\u0000\u0000\u0018��a\u0003]a�Џ�����wM\u0017�\ri��m�y!�(�\u0010��7\u0003
\u0005\u0001\u0000137.74.170.240:5222\n178.170.189.80:5222\n178.20.156.38:5222\n
178.20.157.166:5222\n185.118.166.53:5222\n185.118.66.196:5222\n185.127.24.189:52
22\n185.127.25.195:5222\n185.22.172.134:5222\n188.127.249.36:5222\n193.124.179.9
9:5222\n193.124.180.56:5222\n193.124.180.73:5222\n195.123.210.64:5222\n5.135.76.
16:5222\n88.214.236.11:5222\n91.200.14.80:5222\n91.219.31.14:5222\n91.226.92.206
:5222\u0000\u0000\u0000\u0000\u0000",
        "type": 0,
        "id": 17818585748893203524
    },
    {
        "content": "\u001e�\u0014\u0000",
        "type": 1,
        "id": 15182702438468003372
    }
]
```

Out of a large number of possible resources, the most important are the new peer list (only if its hash differs from current), or announcement of a new DLL being available to download. The latter resource has its own structure for communication purposes (a real matrioshka!), also made of a list of concatenated sub-resources of the following form:

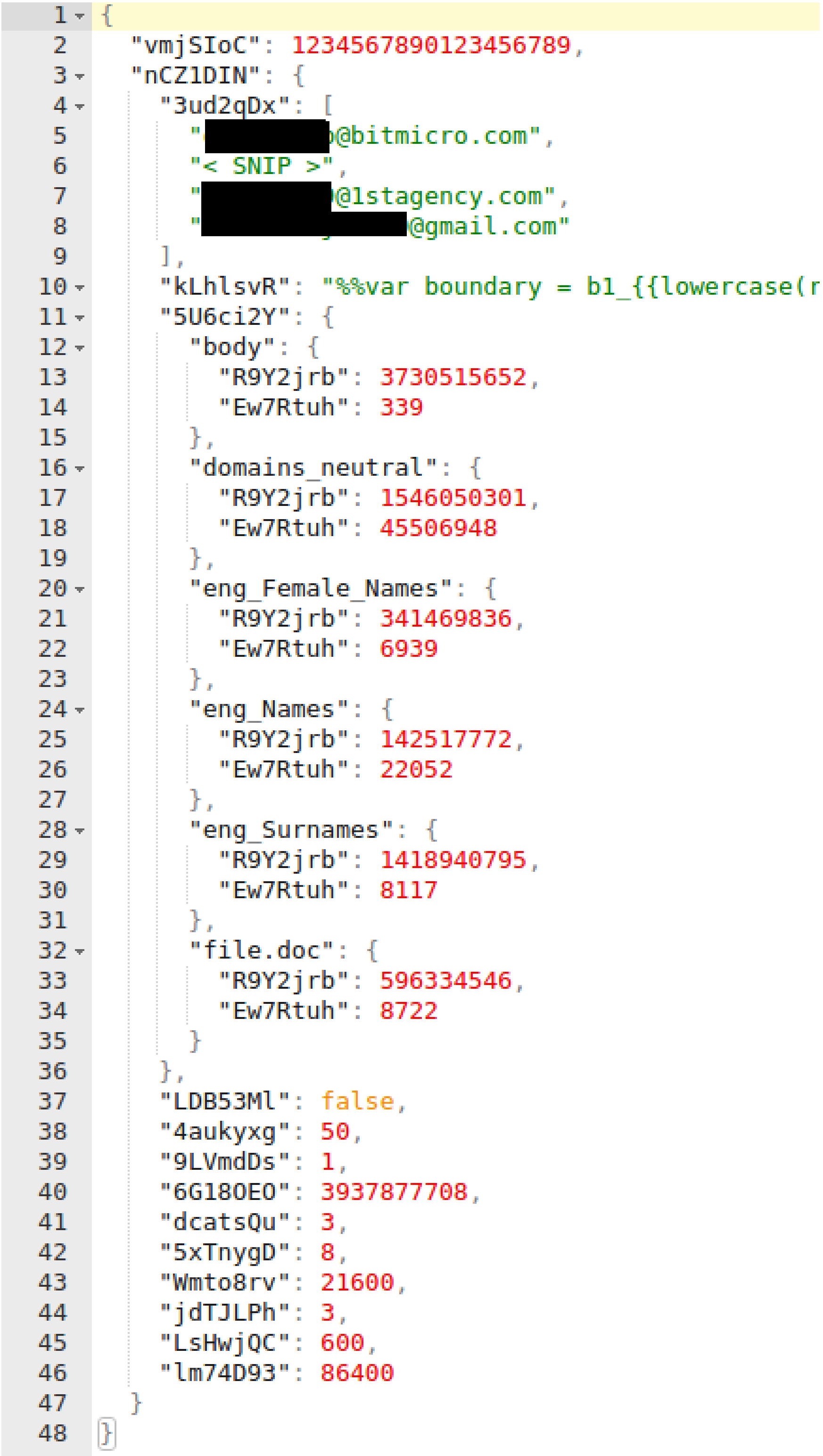
The command can be interpreted as a request for running DLL identified by its SHA-1 with command line parameters stated in *cmdline* field – in practice, the argument is a newline-separated list of C2 addresses (with HTTP path) to be connected to.

Spam C2 communication

The last protocol I will describe in this post, is the communication of the downloaded DLL module, whose responsibility is to send spam emails. The information is wrapped in the following structure (sent as POST data over HTTP):

The encryption algorithm:

After decryption, there are no more steps – we receive raw data as a JSON string (unless the compression flag was set, in which case the data needs to be unpacked – as we found out, a QuickLZ library was used in the malware for this purpose). Unfortunately, keys are obfuscated, so we had to guess their meaning. Sample JSON (pretty-printed and edited to fit on screen):

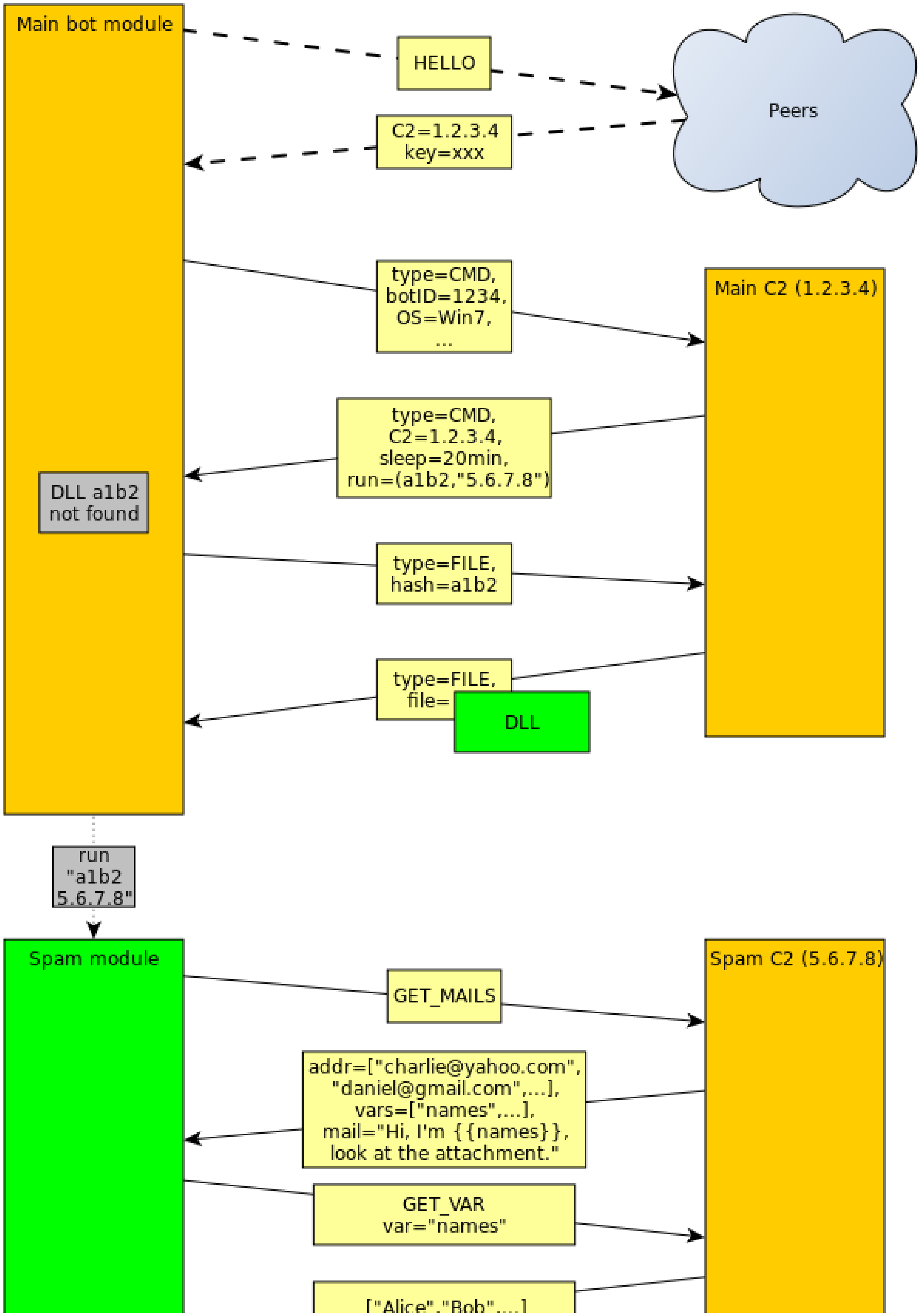


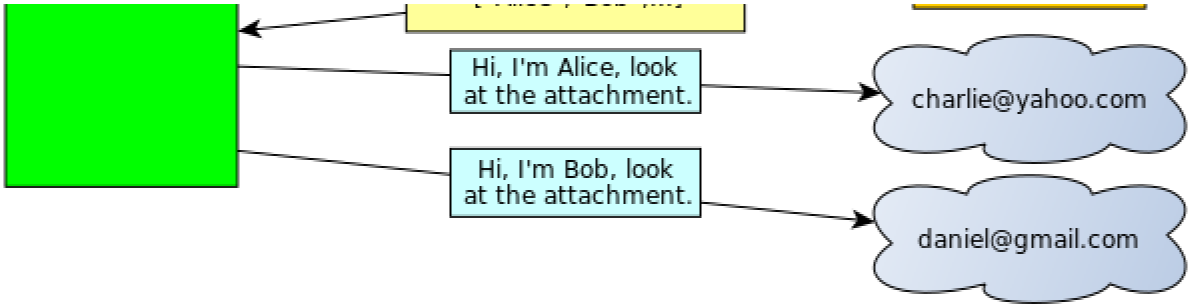
```
{
  "vmjSIoC": 12345678901234567890,
  "nCZ1DIN": {
    "3ud2qDx": [
      "[redacted]@bitmicro.com",
      "< SNIP >",
      "[redacted]@1stagency.com",
      "[redacted]@gmail.com"
    ],
    "kLhlsvR": "%%var boundary = b1_{{lowercase(r
    "5U6ci2Y": {
      "body": {
        "R9Y2jrb": 3730515652,
        "Ew7Rtuh": 339
      },
      "domains_neutral": {
        "R9Y2jrb": 1546050301,
        "Ew7Rtuh": 45506948
      },
      "eng_Female_Names": {
        "R9Y2jrb": 341469836,
        "Ew7Rtuh": 6939
      },
      "eng_Names": {
        "R9Y2jrb": 142517772,
        "Ew7Rtuh": 22052
      },
      "eng_Surnames": {
        "R9Y2jrb": 1418940795,
        "Ew7Rtuh": 8117
      },
      "file.doc": {
        "R9Y2jrb": 596334546,
        "Ew7Rtuh": 8722
      }
    },
    "LDB53Ml": false,
    "4aukyxg": 50,
    "9LVmdDs": 1,
    "6G18OEO": 3937877708,
    "dcatsQu": 3,
    "5xTnygD": 8,
    "Wmto8rv": 21600,
    "jdTJLPh": 3,
    "LsHwjQC": 600,
    "lm74D93": 86400
  }
}
```

Finally, one of the fields in the received dictionary contains a script used to generate randomized emails (like on the top of the post), and as another field – list of parameters passed to this script (e.g. *eng_Names*). We can make a separate request to download value of these arguments – as a response, we will receive, for example, list of English names to be substituted, or a few base64-encoded files to be used as an attachment.

Example communication

I’m aware understanding all those structures and ways they are used is quite hard, so I have created a simplified graph showing the data flow. Example communication could look like this:




Hi, I'm Alice, look at the attachment.
charlie@yahoo.com
Hi, I'm Bob, look at the attachment.
daniel@gmail.com

Sample hashes:
fe929245ee022e3410b22456be10c4f1 - original file (packed)
35be639c5618272f70a0bbfbc25d4465 - dropped DLL module

YARA rules: