SPEAKING DYREZA PROTOCOL. ADVANTAGES OF 'LEARNING' A NEW LANGUAGE

Alexandru Maximciuc & Cristina Vatamanu
Bitdefender, Romania

Email {amaximciuc, cvatamanu}@bitdefender.com

ABSTRACT

Most malware families are capable of evading detection and ensuring long persistence on infected machines through their update mechanisms. However, if one is able to reverse engineer such a sample and simulate C&C communication, invaluable information can be obtained. First, this means we can limit damages caused by the malware by providing near real-time detection, and second the malware's intent can be studied by gathering the configuration files that usually come on the same channel as the other payloads.

In this paper, the steps needed to simulate malware communication traffic are analysed. The paper concentrates on dissecting the network communication, encryption and update mechanisms for one of the most active malware families in 2015, the Dyreza banker. Since the malware distribution is realized across many campaigns, the stages of impersonating various bots with various configurations at the same time in an efficient and scalable way, are also discussed. Using the method described, we have been able to extract important information, such as campaign ID, addresses of the C&C servers, additional modules that are not always downloaded during an update, and, of course, the configuration file that contains all the targeted banks. Besides getting us one step ahead of the malware, this information has helped us gain an insight into the way the botnet is coordinated and divided across different geographic regions.

INTRODUCTION

Malware has evolved over time, but old types of malware still work in tricking the user. Whereas in 2013–2014 the new trend on the malware scene was the controversial 'locker' families, in 2015, one of the most active pieces of malware was the Dyreza banker. Since they appeared, bankers have modified their methods of stealing credentials, adapting to the protection methods adopted by the banks' web servers. Even though neither the web-inject method used by Dyreza to steal credentials, nor its spreading method (via spam campaigns) is new, it seems that they still do the trick.

Although it relies on some old methods, Dyreza is a sophisticated piece of malware. Its network is complex, its communication protocol is complicated, and its update process is divided into many components.

This paper focuses on these aspects, trying to gain an insight into the direction in which the botnet is heading.

DYREZA REVIEW

Dyreza is one of the most important malware families spread in 2015 and it has been widely analysed and reversed. Although many researchers have investigated this piece of malware, let's have a quick recap of its main features.

One of the malware's infection vectors (and the most 'important' one) consists of spam campaigns which deliver the Upatre Downloader. Once on the system, Upatre downloads and executes Dyreza's binary file. Over time, Upatre's payloads changed their encryption method and the download has 'moved' from HTTP to HTTPS in order to reduce its 'visibility' to many protection solutions.

Once decrypted by its downloader, the Dyreza binary file has its own encryption layer. We'll take as an example the file with SHA1 hash 'fd14ff07b1ca08d7beacee08e540703fd71b3181'. After applying a XOR operation to each byte with 0x01, we find another MZ/PE file inside. Its hash is '0861c1c5d1ba2935c3424fefa4c2d2b3c610e6d6'. The encryption layer for this one is based on the VMPC algorithm. The Dyreza binary file hidden under the VMPC decryption is 'fd028de0a84762f3f05ab8c799b82a5071ed985e', which has the resources shown in Figure 1.

Now, of course, these resources are also encrypted – but in this case, it's only a permutation. The last resource, XFNPZPWM1, is actually the permutation table for the first two:

- BTZE393NE – the main Dyreza DLL, for x86 systems
- POZD1F6E2 – the main Dyreza DLL for AMD64 systems.

In this example, we'll go further with '10d2436272ba6b0123d061c4c90926088d7efc5d' (extracted from BTZE393NE after decryptions), which has the resources shown in Figure 2.

» Resources

Type	Name	ID	File Offse	Size	CodePage	Language
RCData	BTZE393NE	0x100	0x3B3C	0x20650	1252	English
RCData	POZD1F6E2	0x114	0x2418C	0x3B020	1252	English
RCData	XFNPZPWM1	0x128	0x5F1AC	0x100	1252	English
Manifest		0x1	0x5F2AC	0x15A	1252	English

Figure 1: Resources for fd028de0a84762f3f05ab8c799b82a5071ed985e.

» Resources

Type	Name	ID	File Offse	Size	CodePage	Language
RCData	0Y2HGIF36	0x160	0x10DC4	0x12EF	1252	English
RCData	4QUYNQKU1	0x174	0x120B4	0xCBEB	1252	English
RCData	6ET5APHF3	0x188	0x1ECA0	0x30	1252	English
RCData	733YSOAC4	0x19C	0x1ECD0	0xA0	1252	English
RCData	9TDUCOGN5	0x1B0	0x1ED70	0x4A0	1252	English
Manifest		0x2	0x1F210	0x15A	1252	English

Figure 2: Resources for 10d2436272ba6b0123d061c4c90926088d7efc5d.

As can be seen in Figure 2, there are five resources in this sample. Four of them are encrypted as follows:

- 0Y2HGIF36 and 4QVYNQKU1 are encrypted with a simple XOR with the first 32 bytes in 6ET5APHF3
- 733YSOAC4 and 9TDUCOGN5 are encrypted with AES256-CBC (which will be described later).

The most important resource for our project is 9TDUCOGN5, which we will refer to as baseConfig from now on.

COMMUNICATION PROTOCOL

All binary files come with embedded encrypted configuration data (baseConfig), which contains, among others, the campaign ID and a list of server IPs to connect to (Figure 3).

Figure 3: Example of decrypted configuration data (baseConfig).

After parsing it, the malware tries to connect, successively, to the IPs specified in the baseConfig in order to retrieve an XML resource. Figure 4 shows the format of the request for this operation.

```
https://baseConfigServerIP/<campaignID>/<botID>/0/<windowsVersion>/
<botVersion>/<botExternalIP>/

Request example:

https://85.192.165.229:443/2602uk21/UDP-
EF5C86498C6 W512600.C2F765D7168392652B390C0E0702D595/0/Win XP 32bit
```

Figure 4: Example request for XML resource.

Where:

- baseConfigServerIP is hard-coded in baseConfig
- campaignID is hard-coded in baseConfig
- botID is a concatenation of:
  - <COMPUTERNAME>
  - "_W"
  - <winMajorVer>
  - <winMinorVer>
  - <winBuildNumber>
  - "."
  - <MD5HEX(<COMPUTERNAME>)>
- windowsVersion is the *Windows* version, e.g. Win_7, Win_XP_32bit, Win_Vista_SP1
- botVersion is hard-coded in the main Dyreza DLL
- botExternalIP is the computer's external IP address; usually, the bot uses legitimate STUN servers to find it.

It looks as if no validation is made server-side regarding the MD5 hash and the computer name.

The above request will retrieve a buffer containing an XML file. The encrypted buffer is shown in Figure 5.

This XML file will contain different server IPs with special roles to which the bot will connect subsequently, sending or retrieving other data.

The format of the response is show in Figure 6.

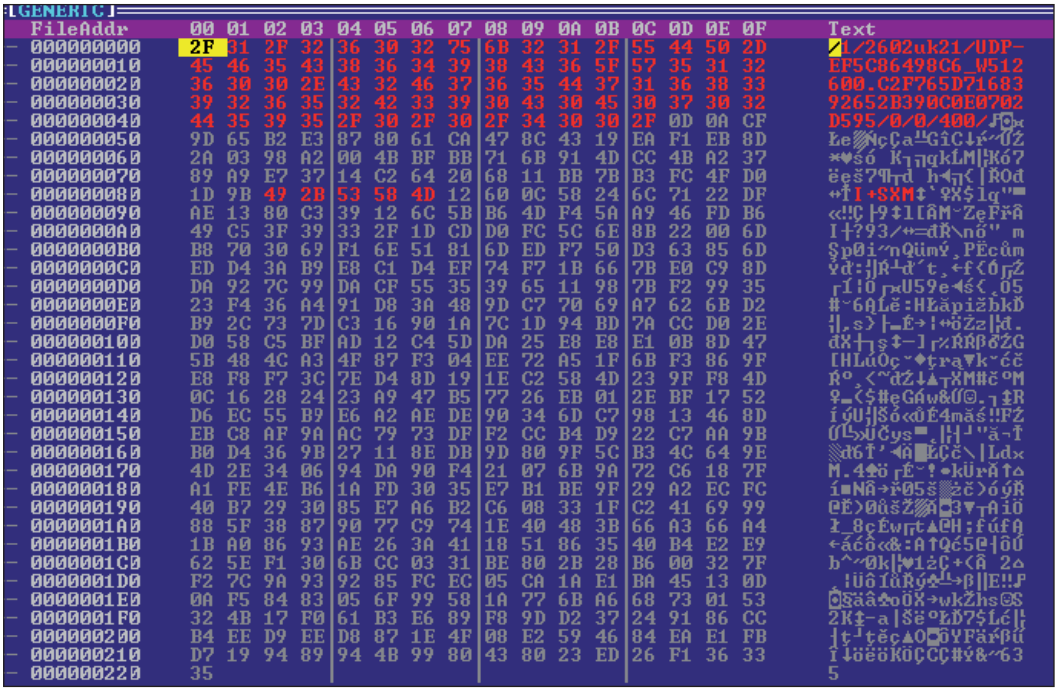

Figure 5: Encrypted buffer containing the XML file.

```
/1/<campaignID>/<botID>/0/0/<encodedDataLength>\x0d\x0a
<encodedData>\x0d\x0a<extraBytes>
```

Figure 6: Response format.

Figure 7 shows the resultant XML file after decrypting the encodedData (the 400 bytes in the above example).

```
<datapost>
<dpsrv>
176.123.16.30:443
</dpsrv>
</datapost>

<modules>
<modsrv>
188.255.154.224:443
</modsrv>
81.162.123.76:443
</modsrv>
199.120.97.190:443
</modsrv>
</modules>

<commands>
<csrv>
1.2.3.4:443
</csrv>
</commands>
```

Figure 7: XML file.

Every server has its own purpose. For example, <modules> servers are used to get the 'plug-ins':

- wg – form-grabber plug-in
- tv, vnc – plug-ins with VNC capabilities
- m_i2p – the I2P communication plug-in.

The encryption algorithm is the same for all the components, embedded or downloaded. It comes in the form of AES256 CBC. The AES key and IV are computed using the SHA256 hash function applied to the first 0x30 bytes of the encrypted buffer:

- the first 0x20 bytes from the buffer are used to generate the key
- the next 0x10 bytes from the buffer are used to derive the IV.

The AES key and IV computation code is illustrated in Figure 8.

As far as we've seen, the cntRounds parameter has two possible values:

- 64 – when the bot wants to decrypt a downloaded resource or the embedded baseConfig
- 1 – when the bot wants to decrypt the 'state file' located on the computer's hard drive.

The Dyreza banker is a very sophisticated and complex piece of malware. For this paper we didn't invest too much time in reversing all the bits in the binaries, but rather we focus on a few important components and the methods used by the bot to keep them up to date.

FRAMEWORK

The primary role of our framework is to monitor Dyreza's network and the update of its configuration files and to help us understand its dimension and geographic distribution.

Figure 9 provides a summary of how the framework works.

The framework is subscribed to the Dyreza collection. When a new binary file is encountered, it is first unpacked and then the embedded information is extracted (baseConfig and botVersion). These pieces of information are inserted into our database for follow-up correlations.

The next step is the impersonation of a valid zombie. We have to randomly generate values for bot ID, computer name, external IP and *Windows* version in order to build the request for the XML resource (Figure 4). If the download succeeds, the received buffer is decrypted and parsed and kept internal for the bot instance. In the XML file we have a list of modules, datapost and commands servers. If, on the other hand, the download or decryption fails at some point, we retry it with a different C&C address from the baseConfig file (we limited the retry count to 33, which is usually a little more than a half of the C&C servers specified in the baseConfig file – the bad guys are pretty generous!)

Parsing the XML resources, new IP servers are retrieved, some of which are used later to fetch the plug-ins, while others are only flagged in our database. The new request for these plug-ins is shown in Figure 10.

```
// key derivation function helper
func KDF_sha256(buffer []byte, cntRounds, sizeBufferIn, sizeBufferOut int) []byte {
	bufferOut := computeSHA256(buffer[0 : sizeBufferIn])
	bufferTmp := make([]byte, 0x30)
	if cntRounds > 0 {
		for i := 0; i < cntRounds; i++ {
			if copy(bufferTmp, bufferOut) != 0x20 {
				panic("ERROR copying bytes!")
			}
			for j := 0; j < 0x10; j++ {
				bufferTmp[0x20 + j] = bufferOut[j] + 1
			}
			bufferOut = computeSHA256(bufferTmp)
		}
	}
	return bufferOut[0 : sizeBufferOut]
}

// key derivation function
func KDF(buffer []byte, cntRounds int) (key, iv []byte) {
	key = KDF_sha256(buffer[0 : 32], 2 * cntRounds, 32, 32)
	iv = KDF_sha256(buffer[32 : 32 + 16], cntRounds, 16, 16)
	return
}
```

Figure 8: Decryption code.

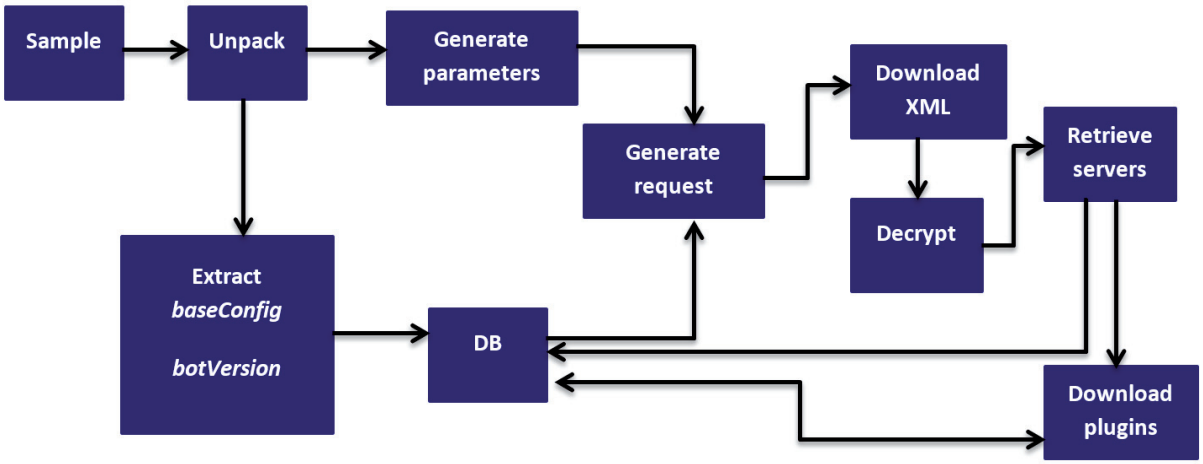

Figure 9: Framework's flow.

```
https://serverIP/<campaignID>/<botID>/5/<componentName>/<botExternalIP>
```

Figure 10: Request for plug-in download.

The componentNames of interest to us at the moment are:

- respparser, httprex, httprex2, respparser2, rps2 and bccfg – the serverIP in this case is from the C&C list from the baseConfig resource.
- tv32, wg32, m_i2p32, vnc32, tv64, wg64, m_i2p64 and vnc64 – in this case the serverIP is from the modules list in the previously decrypted XML resource.

If the whole process succeeds, the downloaded buffer is decrypted. The hashes (SHA512) for encrypted and decrypted buffers are stored in the database for further correlations.

The final step is to retrieve a new update for the baseConfig resource (newBaseConfig). This newBaseConfig will replace the old baseConfig at the next iteration in our framework. The request is in the format shown in Figure 11.

The response illustrated in Figure 12 contains, besides the campaignID and botID, the new version for the baseConfig resource.

Once the decryption process has successfully been completed, information from the new configuration file is inserted into the database (the IPs for the new servers). If the component is

```
https://baseConfigServerIP/<campaignID>/<botID>/23/<current baseConfig
version>/<random string of length 31>/<botExternalIP>

Example:
https://195.206.255.131/man1/DW7-
PC_W617601.A6AE9B95DC5145987680008B16FC584B1/23/230194640/WTtEwJvbHIysbNg
KYgSJaMyDkYPEnaE/176.126.252.11/
```

Figure 11: Request for newBaseConfig.

Figure 12: Encrypted newBaseConfig.

not known to us (the computed SHA512 hash on the decrypted buffer is new), a notification is sent.

Should the decryption fail, we save the raw buffer for further inspection and send a notification of failure.

After all the servers have been used for downloading new data, the whole process reiterates, now using the newly added servers' IPs from the database alongside the old ones in the download processes.

ADVANTAGES AND DISADVANTAGES

The main advantage of this project is its scalability: with a single machine you can 'pretend' to have hundreds of infected machines and get a better insight into the payloads, or you could bypass any 'sleeps' imposed by the malware in a normal infection scenario.

Another important advantage is that the framework is capable of requesting a certain resource that would be served only in special circumstances by simulating every necessary condition.

We chose to write this project in Go (golang) because of its built-in concurrency (and we use it a lot, running about 30 'infected machines' at the same time), C-resemblance, static typing and static linking. Also, it's a nice language to play with.

The main disadvantage of a project like this is that one has to invest a lot of time in reversing and re-building the protocol in a language of your choice, but after finalizing the project the results are worth it.

STATISTICS

In our four months of investigations we processed approximately 3,000 samples. At the time of the writing this paper, we have registered 242 different campaign IDs in our database. Most of them have a standard format, a concatenation between a date (day and month), a country id and a number (2402uk2, 0903us23, 2402uk1, 2502uk1, 1903no13). There are two exceptions among the campaign IDs: man and cor. These appear to be accompanied only by numbers: man1, man2, man3, man4 and cor1.

Analysing our data, we didn't find a certain campaign that would target a particular country or a particular bank. The resources in charge of defining the redirection from the legitimate URL to the malicious domain server seem to have almost the same list of banks (or targeted sites) among all the campaigns. From time to time, small updates are made, adding new web pages to the existing list of 'victims'. Also, we observed that, over time, the malware creators added new types of 'victims'. While at first the list of URLs represented only banking institutions and financial groups, recently updates have also contained payment services, shopping websites, sites that sell or buy bitcoins, domain registration, mail-sender and web-hosting services, job marketplaces and others.

At the time of the writing this paper, we had extracted 585 targeted websites from the downloaded resources. The most affected countries, in regards to banks or financial institutions, are illustrated in Figure 13.

As can be seen, the countries with the biggest number of targeted institutions are the United Kingdom with 79, Germany with 59, and Australia with 48 financial institutions.

Another important aspect we observed is that the IPs for any of the servers change often (new IPs appear in our database weekly). 802 distinct IP addresses (here we include the servers from the baseConfig files and the servers contained in all the downloaded resources) passed through our system in four months. It seems that most of them are (were) located in Ukraine and Russia, as can be seen in Figure 14.

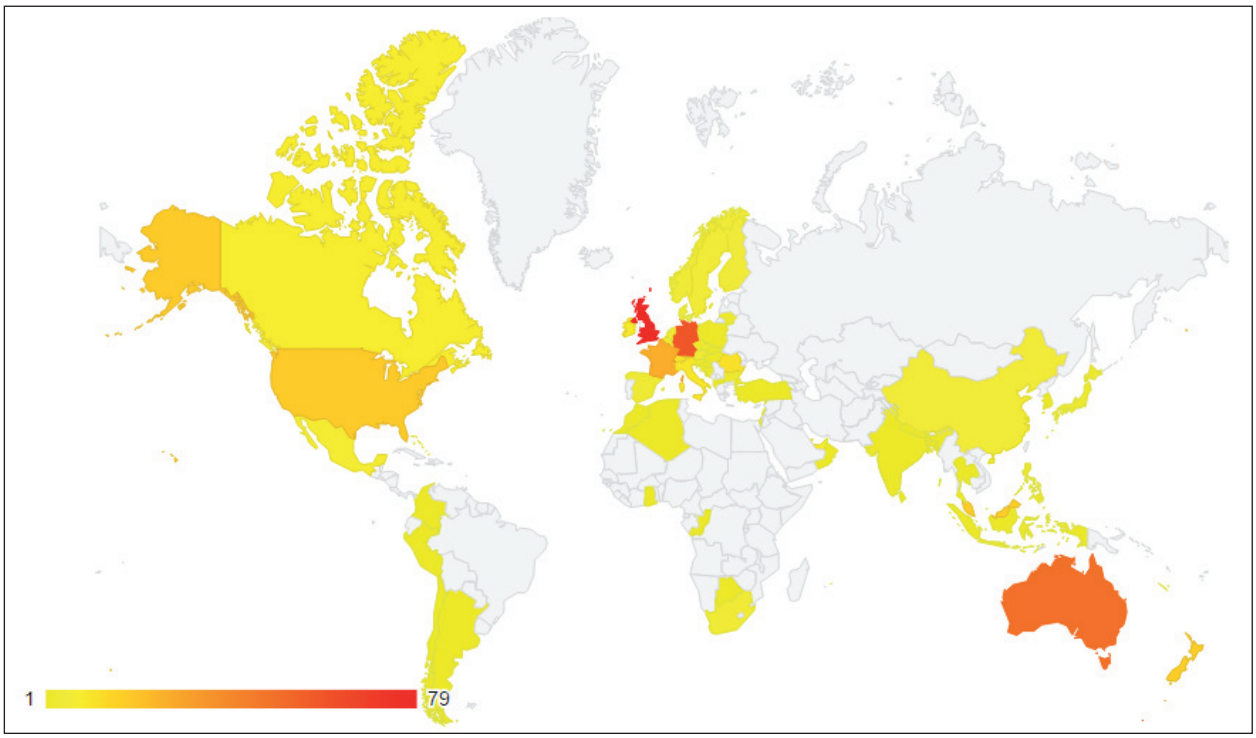

Figure 13: Top targeted countries.

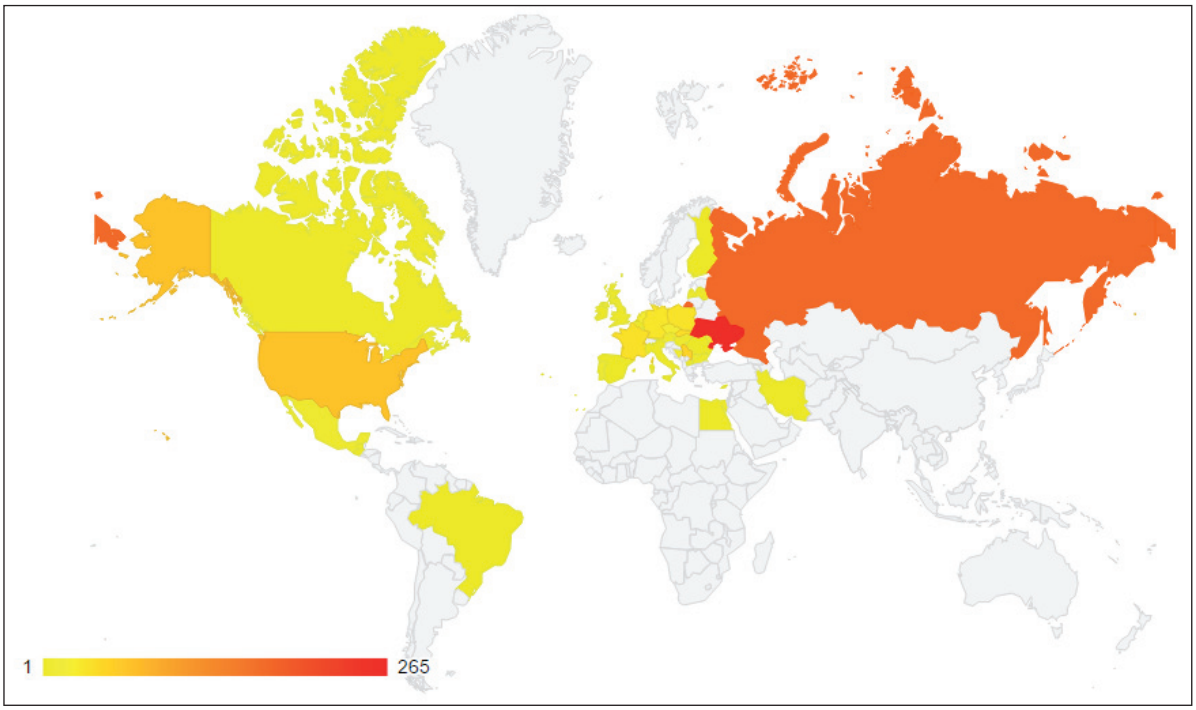

Figure 14: Dyreza servers' geographical distribution.

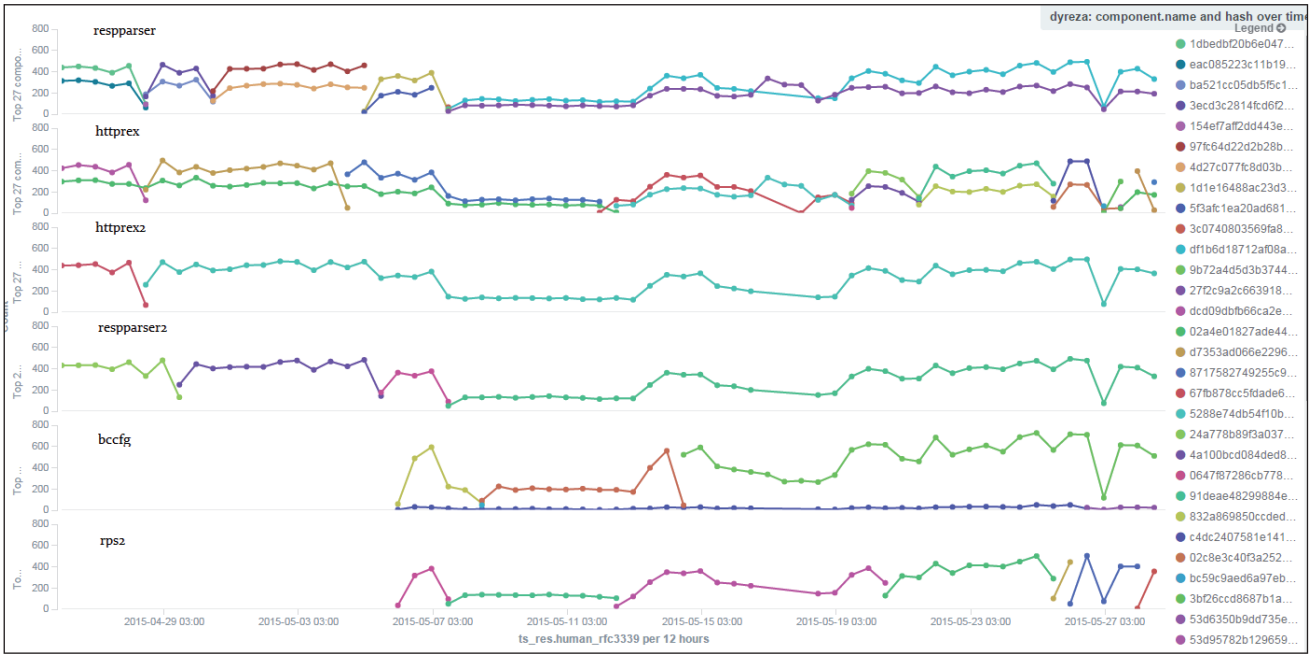

Figure 15: Update process.

Another interesting thing was to follow the update process for the downloaded resources. Figure 15 illustrates the updates for the respparser, httprex, httprex2, respparser2, bccfg and rps2 components. The most intriguing aspect retrieved from our database is that there seem to be two different configurations running at the same time for some of the resources, specifically for the respparser, httprex and bccfg components. The graphic illustrates data between 28 April and 27 May.

Let's take for example the respparser component. As can be seen, there are two streams of updates running for this component at the same time. Both streams were changed on 28 and 30 April, 5 May and 7 May.

The differences were as follows:

- On 28 April:
  - 'stream1': 1dbedbf20… changed to 3ecd3c2814f… The only major change was the server used for redirects.
  - 'stream2': eac085223c.. changed to ba521cc05db5f5c1e96.. The only major change was the server used for redirects.

The difference between the new resources (3ecd3c2814f.. and ba521cc05db5f5c1e96..) refers to the targeted sites and consists of four modified lines.

- On 5 May:
  - 'stream1': 97fc64d22d2b.. changed to 1d1e16488ac23d3a.. Updates: four modified URL-parts and one IP (used for redirects).
  - 'stream2': 4d27c077fc8d03.. changed to 5f3afc1ea20ad681c9.. Updates: six modified URL-parts and one IP (used for redirects).

    The interesting part is that the only difference between the new resources (1d1e16488ac23d3a and 5f3afc1ea20ad681c9) is now an IP address.

At the time of writing this paper, both respparser 'streams' contain two IP addresses (one that is shared between the two streams and one that is different) and they both have the same list of URL-parts they are interested in.

The httprex resource, as can be seen above, suffered an update on one of the streams on 28 April, specifically from dcd09dbfb66ca2e17b.. to d7353ad066e22969..; the change consisted of:

- A server address change
- The addition of a few new URLs of interest and the 'repair' of some of the old targeted URLs.

On 4 May, that same 'stream' changed again (from d7353ad066e22969.. to 8717582749255c91a..), again by changing the IP address of the server (see Figure 16); also, they added 10 more 'targets' to the configuration file.

Around 12 May, the same stream got an update again (this time only the server address was changed), which was followed shortly afterwards by an update of the second stream (from 02a4e01827ade443.. to 5288e74db54f10ba6..):

- A server address change
- Lots of targeted domains were added.

Some of these new targeted domains were added to the first stream on 28 April.

Another interesting thing is that most of the campaigns we're impersonating are tied to a specific stream, but there are a few campaigns that from time to time do a 'stream-boundary-trespassing'. For example, Figure 17 plots data from 21 May to 25 May, for respparser (above) and httprex (below). The x axis represents all the campaigns we follow. The y axis represents the number of successful downloads for the component over that period of time.

There were a few bots corresponding to specific campaigns that, over that period, mostly fetched resources for one of the streams and on a few occasions fetched resources for the other stream (for the respparser resource the first stream is shown in light green and the second stream in dark green, and for the httprex resource the first stream is shown in light purple and the second stream in dark purple). If we were to zoom into the image we would see that some of the campaigns that made those 'stream-boundary-trespasses' were: 0204us22, 1102us2 and 1902us1 (see Figure 18).

CONCLUSIONS

'Learning a new language' takes time and you always have to keep an eye on fresh samples and validate that the protocol and the resources are still the same – but once all of this is done, some interesting aspects are highlighted. Speaking the same protocol as Dyreza brought us new insights into the botnet. Although it seemed at first to be 'just another banker', we learned by retrieving components that are not

Figure 16: Streams differences for the httprex resource.

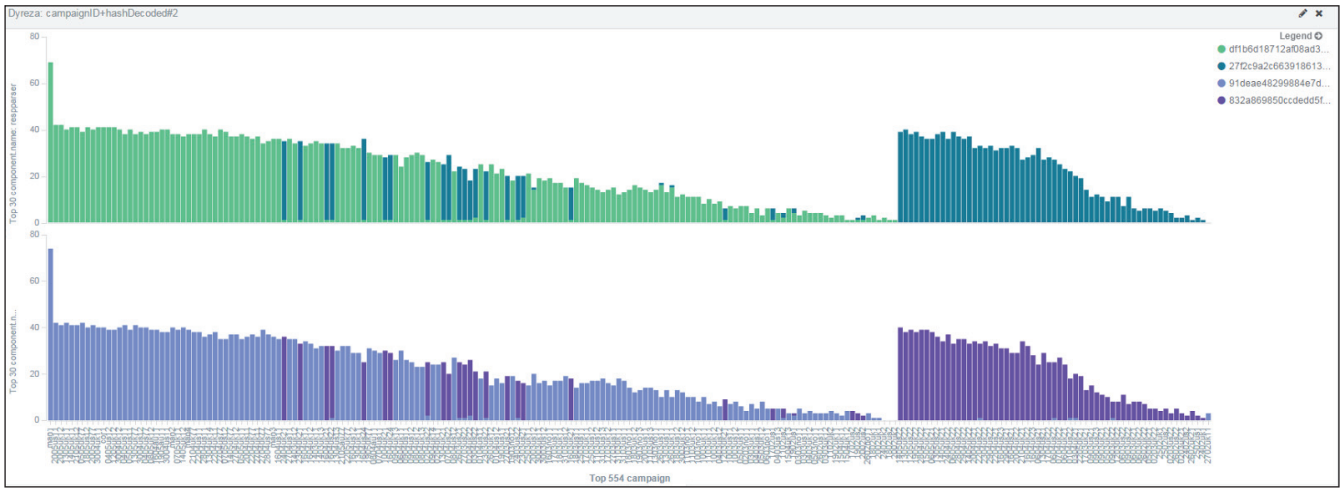

Figure 17: Campaigns' stream-transitions.

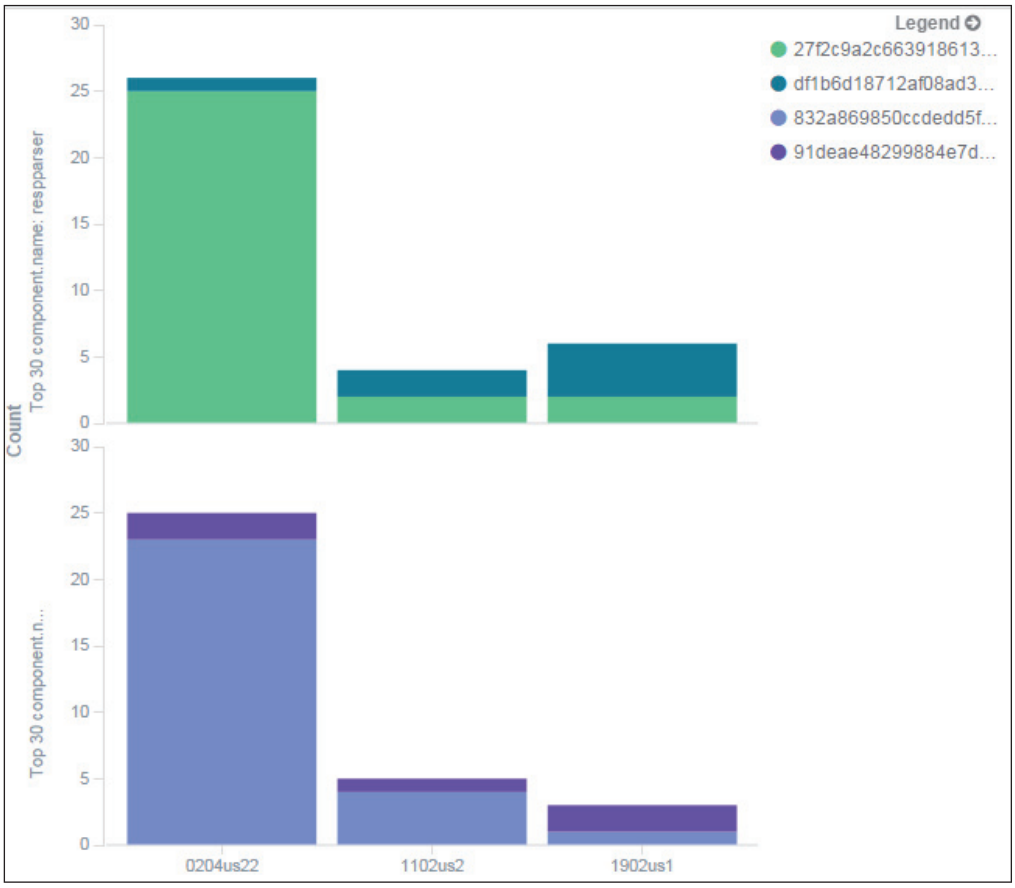

Figure 18: Stream-transition for campaigns 0204us22, 1102us2 and 1902us1.

downloaded, or which take a long time before being downloaded in a normal infection, that this is a complex piece of malware. We were able to impersonate many infections for different campaigns in a scalable manner. Based on this information we saw how the botnet is coordinated and divided across different geographic regions and how the update process is carried out between different campaigns over time.

ACKNOWLEDGEMENTS

This work was co-funded by the European Social Fund through Sectoral Operational Programme Human Resources Development 2007 – 2013, project number POSDRU/187/1.5/S/155397, project title 'Towards a New Generation of Elite Researchers through Doctoral Scolarships'.