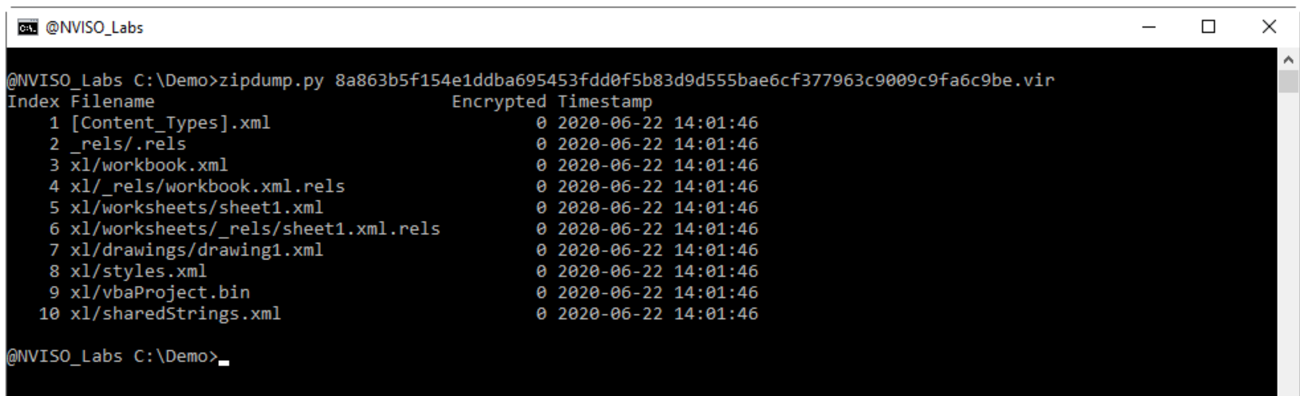
Epic Manchego – atypical maldoc delivery brings flurry of infostealers

blog.nviso.eu/2020/09/01/epic-manchego-atypical-maldoc-delivery-brings-flurry-of-infostealers/

September 1, 2020



In July 2020, NVISO detected a set of malicious Excel documents, also known as “maldocs”, that deliver malware through VBA-activated spreadsheets. While the malicious VBA code and the dropped payloads were something we had seen before, it was the specific way in which the Excel documents themselves were created that caught our attention.

The creators of the malicious Excel documents used a technique that allows them to create macro-laden Excel workbooks, without actually using Microsoft Office. As a side effect of this particular way of working, the detection rate for these documents is typically lower than for standard maldocs.

This blog post provides an overview of how these malicious documents came to be. In addition, it briefly describes the observed payloads and finally closes with recommendations as well as indicators of compromise to help defend your organization from such attacks.

Key Findings (TL;DR)

- The malicious Microsoft Office documents are created using the EPPlus software rather than Microsoft Office Excel, these documents may fly under the radar as it differs from a typical Excel document;
- NVISO assesses with medium confidence that this campaign is delivered by a single threat actor based on the limited number of documents uploaded to services such as VirusTotal, and the similarities in payloads delivery throughout this campaign;
- The payloads that have been observed up to the date of the release of this post, have been, for the most part, so called information stealers with the intention of harvesting passwords from browsers, email clients, etc.;

- The payloads stemming from these documents have evolved only slightly in terms of obfuscation and masquerading. This is another indication of a single actor who is slowly evolving their technical prowess.

Analysis

The analysis section below is divided in two parts and refers to a specific link in the infection chain.

Malicious document analysis

In an earlier blog post, we wrote about "VBA Purging"[1], which is a technique to remove compiled VBA code from VBA projects. We were interested to see if any malicious documents found in-the-wild were adopting this technique (it lowers the initial detection rate of antivirus products). This is how we stumbled upon a set of peculiar malicious documents.

At first, we thought they were created with Excel, and were then VBA purged. But closer examination leads us to believe that these documents are created with a .NET library that creates Office Open XML (OOXML) spreadsheets. As stated in our VBA Purging blog post, Office documents can also lack compiled VBA code when they are created with tools that are totally independent from Microsoft Office. EPPlus is such a tool. We are familiar with this .NET library, as we have been using it since a couple of years to create malicious documents ("maldocs") for our red team and penetration testers.

When we noticed that the maldocs had no compiled code, and were also missing Office metadata, we quickly thought about EPPlus. This library also creates OOXML files without compiled VBA code and without Office metadata.

The OOXML file format is an Open Packaging Conventions (OPC) format: a ZIP container with mainly XML files, and possibly binary files (like pictures). It was first introduced by Microsoft with the release of Office 2007. OOXML spreadsheets use extension .xlsx and .xlsm (for spreadsheets with macros).

When a VBA project is created with EPPlus, it does not contain compiled VBA code. EPPlus has no methods to create compiled code: the algorithms to create compiled VBA code are proprietary to Microsoft.

The very first malicious document we detected was created on 22nd of June 2020, and since then 200+ malicious documents were found over a period of 2 months. The actor has increased their activity in the last weeks, as now we see more than 10 new malicious documents on some days.

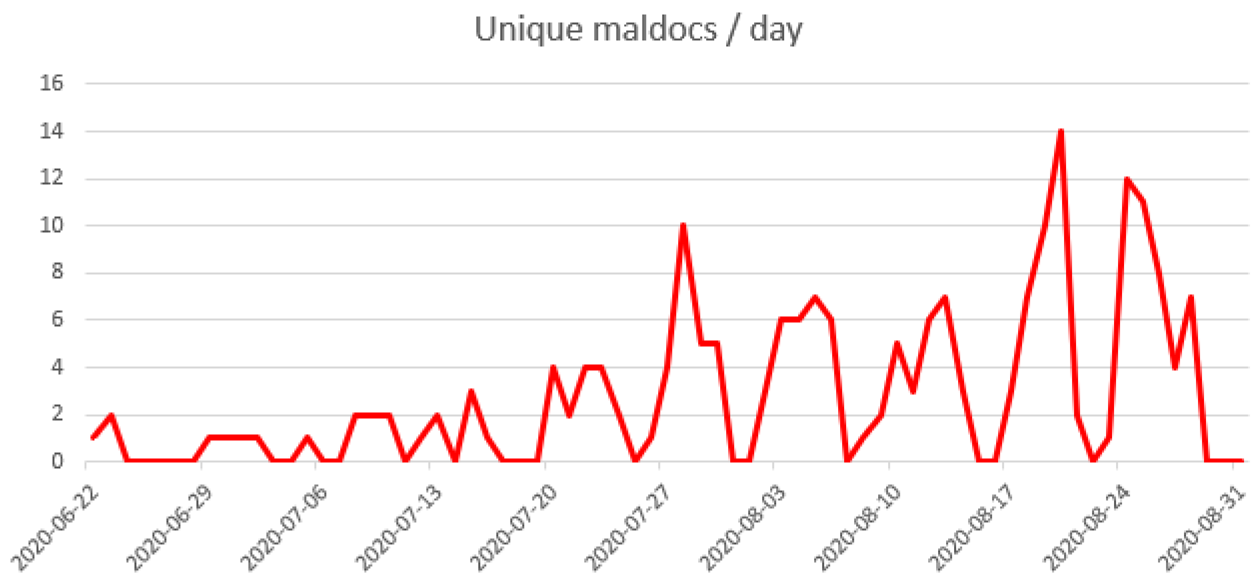


Figure 1 – Unique maldocs observed per day

The maldocs discovered over the course of two months have many properties that are quite different from the properties of documents created with Microsoft Office. We believe this is the case because they are created with a tool independent from Microsoft Excel. Although we don't have a copy of the exact tool used by the threat actor to create these malicious documents, the malicious documents created by this tool have many properties that convince us that they were created with the aforementioned EPPlus software.

Some of EPPlus' properties include, but are not limited to:

- Powerful and versatile library: not only can it create spreadsheets containing a VBA project, but that project can also be password protected and/or digitally signed. It does not rely on Microsoft Office. It can also run on Mono (cross platform, open-source .NET).
- OOXML files created with EPPlus have some properties that distinguish them from OOXML files created with Excel. Here is an overview:

    ZIP Date: every file included in a ZIP file has a timestamp (DOSDATE and DOSTIME field in the ZIPFILE record). For documents created (or edited) with Microsoft Office, this timestamp is always 1980-01-01 00:00:00 (0x0021 for DOSDATE and 0x0000 for DOSTIME). OOXML files created with EPPlus have a timestamp that corresponds to the creation time of the document. Usually, that timestamp is the same for all files inside the OOXML files, but due to execution delays, there can be a difference of 2 seconds between timestamp. 2 seconds is the resolution of the DOSTIME format.

Figure 2 – DOSTIME difference (left: EPPlus created file)

Extra ZIP records: a typical ZIP file is composed of ZIP file records (magic 50 4B 03 04) with metadata for the file, and the (compressed) file content. Then there are ZIP directory entries (magic 50 4B 01 02) followed by a ZIP end-of-directory record (magic 50 4B 05 06). Microsoft Office creates OOXML files containing these 3 ZIP record types. EPPlus creates OOXML files containing 4 ZIP records: it also includes a ZIP data description record (magic 50 4B 07 08) after each ZIP file record.

Figure 3 – Extra ZIP records (left: EPPlus created file)

Missing Office document metadata: an OOXML document created with Microsoft Office contains metadata (author, title, …). This metadata is stored inside XML files found inside the docProps folder. By default, documents created with EPPlus don't have metadata: there is no docProps folder inside the ZIP container.

Figure 4 – Missing metadata (left: EPPlus created file)

VBA Purged: OOXML files with a VBA project created with Microsoft Office contain an OLE file (vbaProject.bin) with streams containing the compiled VBA code and the compressed VBA source code. Documents created with EPPlus do not contain compiled VBA code, only compressed VBA source code. This means that:

- The module streams only contain compressed VBA code
- There are no SRP streams (SRP streams contain implementation-specific and version-dependent compile code, theire name starts with __SRP_)
- The _VBA_PROJECT stream does not contain compiled VBA code. In fact, the content of the _VBA_PROJECT stream is hardcoded in the EPPlus source code: it's always CC 61 FF FF 00 00 00.

Figure 5 – Hardcoded stream content (left: EPPlus created file)

In addition to the above, we have also observed some properties of the VBA source code that hints at the use of a creation tool based on a library like EPPlus.

There are a couple of variants to the VBA source code used by the actor (some variants use PowerShell to download the payload, others use pure VBA code). But all these variants contain a call to a loader function with one argument, a string with the URL (either BASE64 or hexadecimal encoded). Like this (hexadecimal example):

```
Loader”68 74 74 70 …”
```

Do note that there is no space character between the function name and the argument: there is no space between Loader and ”68 74 74 70 …”.

This is an indication that the VBA code was not entered through the VBA EDI in Office: when you type a statement like this, without space character, the VBA EDI will automatically add a space character for you (even if you copy/paste the code). The absence of this space character divulges that this code was not entered through the VBA EDI, but likely via a library such as EPPlus.

To illustrate these differences in properties, we show examples with one of our internal tools (ExcelVBA) using the EPPlus library. We create a vba.xlsm file with the vba code in text file vba.txt using our tool ExcelVBA, and show some of its properties.:

@NVISO_Labs

```
@NVISO_Labs C:\Demo>ExcelVBA.exe --help
ExcelVBA 1.0.0.0
Copyright c  2018

  -p, --password           (Default: ) Encrypt with provided password.

  -V, --VelvetSweatshop    (Default: false) Encrypt with default password.

  -M, --module             (Default: Module1) Module name.

  -S, --sheet              (Default: Sheet1) Sheet name.

  --help                   Display this help screen.

  --version                Display version information.

  vbafile (pos. 0)         Required. File with VBA code.

  excelfile (pos. 1)       Required. Excel file to create.


@NVISO_Labs C:\Demo>type vba.txt
Sub Demo
        Msgbox"Hello"
End Sub

@NVISO_Labs C:\Demo>time
The current time is: 23:13:03.11
Enter the new time:

@NVISO_Labs C:\Demo>ExcelVBA.exe vba.txt vba.xlsm

@NVISO_Labs C:\Demo>zipdump.py vba.xlsm
Index Filename                   Encrypted Timestamp
    1 [Content_Types].xml                0 2020-08-27 23:13:08
    2 _rels/.rels                        0 2020-08-27 23:13:08
    3 xl/workbook.xml                    0 2020-08-27 23:13:08
    4 xl/_rels/workbook.xml.rels         0 2020-08-27 23:13:08
    5 xl/worksheets/sheet1.xml           0 2020-08-27 23:13:08
    6 xl/styles.xml                      0 2020-08-27 23:13:08
    7 xl/vbaProject.bin                  0 2020-08-27 23:13:08
    8 xl/sharedStrings.xml               0 2020-08-27 23:13:08

@NVISO_Labs C:\Demo>oledump.py -i vba.xlsm
A: xl/vbaProject.bin
 A1:       410            'PROJECT'
 A2:        47            'PROJECTwm'
 A3: M     213      0+213 'VBA/Module1'
 A4: m     169      0+169 'VBA/Sheet1'
 A5:         7            'VBA/_VBA_PROJECT'
 A6:       197            'VBA/dir'

@NVISO_Labs C:\Demo>
```

Figure 6 – NVISO created XLSM file using the EPPlus library

```
@NVISO_Labs

@NVISO_Labs C:\Demo>oledump.py -s A3 --vbadecompressskipattributes vba.xlsm
Sub Demo
        Msgbox"Hello"
End Sub

@NVISO_Labs C:\Demo>oledump.py -s A5 vba.xlsm
00000000: CC 61 FF FF 00 00 00                              .a.....

@NVISO_Labs C:\Demo>
```

Figure 7 – Running oledump.py reveals this document was created using the EPPlus library

Some of the malicious documents contain objects that clearly have been created with EPPlus, using some of the example code found on the EPPlus Wiki. We illustrate this with the following example (the first document in this campaign):

Filename: Scan Order List.xlsm
MD5: 8857fae198acd87f7581c7ef7227c34d
SHA256: 8a863b5f154e1ddba695453fdd0f5b83d9d555bae6cf377963c9009c9fa6c9be
File Size: 5.77 KB (5911 bytes)
Earliest Contents Modification: 2020-06-22 14:01:46

This document contains a drawing1.xml object (a rounded rectangle) with this name: name="VBASampleRect".

```
@NVISO_Labs

@NVISO_Labs C:\Demo>zipdump.py 8a863b5f154e1ddba695453fdd0f5b83d9d555bae6cf377963c9009c9fa6c9be.vir
Index Filename                            Encrypted Timestamp
    1 [Content_Types].xml                         0 2020-06-22 14:01:46
    2 _rels/.rels                                 0 2020-06-22 14:01:46
    3 xl/workbook.xml                             0 2020-06-22 14:01:46
    4 xl/_rels/workbook.xml.rels                  0 2020-06-22 14:01:46
    5 xl/worksheets/sheet1.xml                    0 2020-06-22 14:01:46
    6 xl/worksheets/_rels/sheet1.xml.rels         0 2020-06-22 14:01:46
    7 xl/drawings/drawing1.xml                    0 2020-06-22 14:01:46
    8 xl/styles.xml                               0 2020-06-22 14:01:46
    9 xl/vbaProject.bin                           0 2020-06-22 14:01:46
   10 xl/sharedStrings.xml                        0 2020-06-22 14:01:46

@NVISO_Labs C:\Demo>_
```

Figure 8 – zipdump of maldoc

```
@NVISO_Labs C:\Demo>zipdump.py -s 7 -d 8a863b5f154e1ddba695453fdd0f5b83d9d555bae6cf377963c9009c9fa6c9be.vir | grep -C 10
 VBASampleRect
      <xdr:rowOff>0</xdr:rowOff>
    </xdr:from>
    <xdr:to>
      <xdr:col>10</xdr:col>
      <xdr:colOff>0</xdr:colOff>
      <xdr:row>10</xdr:row>
      <xdr:rowOff>0</xdr:rowOff>
    </xdr:to>
    <xdr:sp macro="" textlink="">
      <xdr:nvSpPr>
        <xdr:cNvPr id="0" name="VBASampleRect" />
        <xdr:cNvSpPr />
      </xdr:nvSpPr>
      <xdr:spPr>
        <a:prstGeom prst="roundRect">
          <a:avLst />
        </a:prstGeom>
      </xdr:spPr>
      <xdr:style>
        <a:lnRef idx="2">
          <a:schemeClr val="accent1">

@NVISO_Labs C:\Demo>
```

Figure 9 – Selecting the drawing1.xml object reveals the name

This was created with sample code found on the EPPlus Wiki[2]:

```
private static void VBASample1(DirectoryInfo outputDir)
{
    ExcelPackage pck = new ExcelPackage();
    //Add a worksheet.
    var ws=pck.Workbook.Worksheets.Add("VBA Sample");
    ws.Drawings.AddShape("VBASampleRect", eShapeStyle.RoundRect);
    //Create a vba project
    pck.Workbook.CreateVBAProject();
    //Now add some code to update the text of the shape...
    var sb = new StringBuilder();
    sb.AppendLine("Private Sub Workbook_Open()");
    sb.AppendLine("    [VBA Sample].Shapes(\"VBASampleRect\").TextEffect.Text = \"This text is set from VBA!\"");
    sb.AppendLine("End Sub");
    pck.Workbook.CodeModule.Code = sb.ToString();

    //And Save as xlsm
    pck.SaveAs(new FileInfo(outputDir.FullName + @"\sample15-1.xlsm"));
}
```

Figure 10 – EPPlus sample code, clearly showing the similarities

Noteworthy is that all maldocs we observed have their VBA project protected with a password. It is interesting to note that the VBA code itself is not encoded/encrypted, it is stored in cleartext (although compressed) [3]. When a document with a password protected

VBA project is opened, the VBA macros will execute without the password: the user does not need to provide the password. The password is only required to view the VBA project inside the VBA IDE (Integrated Development Environment):

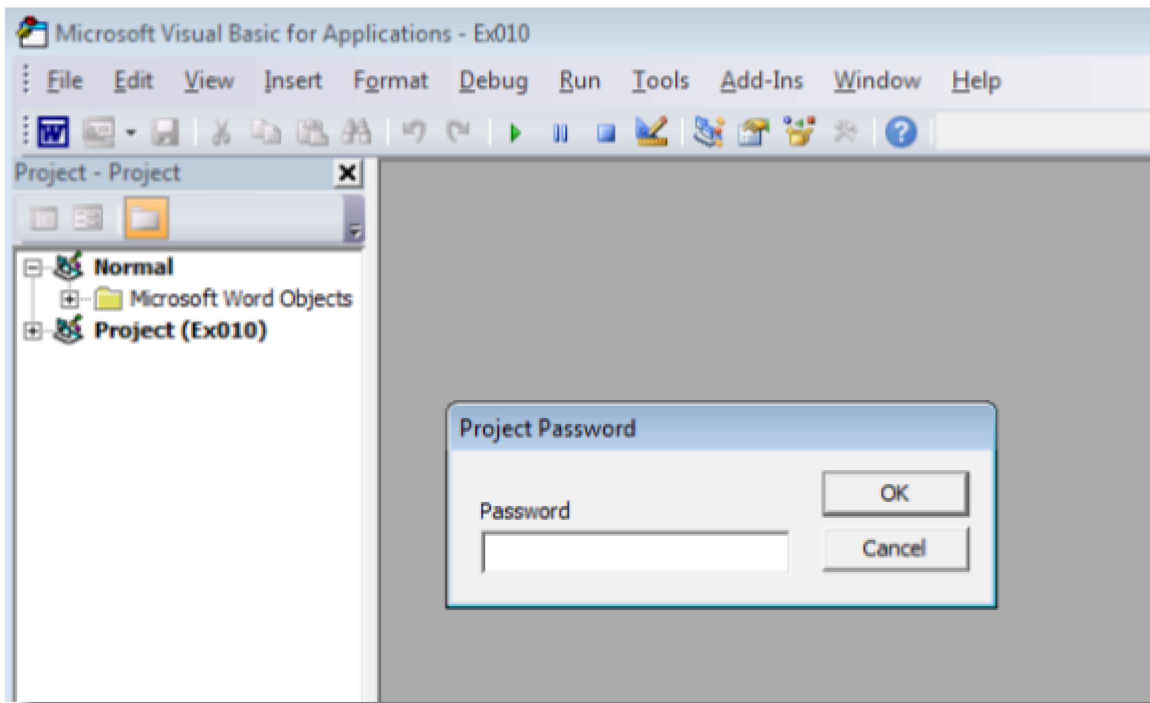


Figure 11 – Password prompt for viewing the VBA project

We were not able to recover these passwords. We used John the Ripper with the rockyou.txt password list[4], and Hashcat with a small ASCII brute-force attack.

Although each malicious document is unique with its own VBA code, with more than 200 samples analyzed to date, we can generalize and abstract all this VBA code to just a handful of templates. The VBA code will either use PowerShell or ActiveX objects to download the payload. The different strings are encoded using either hexadecimal, BASE64 or XOR-encoding; or a combination of these encodings. A Yara rule to detect these maldocs is provided at the end of this blog post for identification and detection purposes.

Payload analysis

As mentioned in the previous section, via the malicious VBA code, a second-stage payload is downloaded from various websites. Each second-stage executable created by its respective malicious document acts as dropper for the final payload. In order to thwart detection mechanisms such as antivirus solutions, a variety of obfuscation techniques are leveraged which are however not advanced enough to hide the malicious intent. The infrastructure used by the threat actor appears to mainly comprise compromised websites.

Popular antivirus solutions such as those listed on VirusTotal, shown in Figure 12, commonly identify the second-stage executables as “AgentTesla”. While leveraging VirusTotal for malware identification is not an ideal method, it does display how simple obfuscation can result in an incorrect classification. Throughout this analysis, we’ll explain how only few of these popular detections turned out to be accurate.

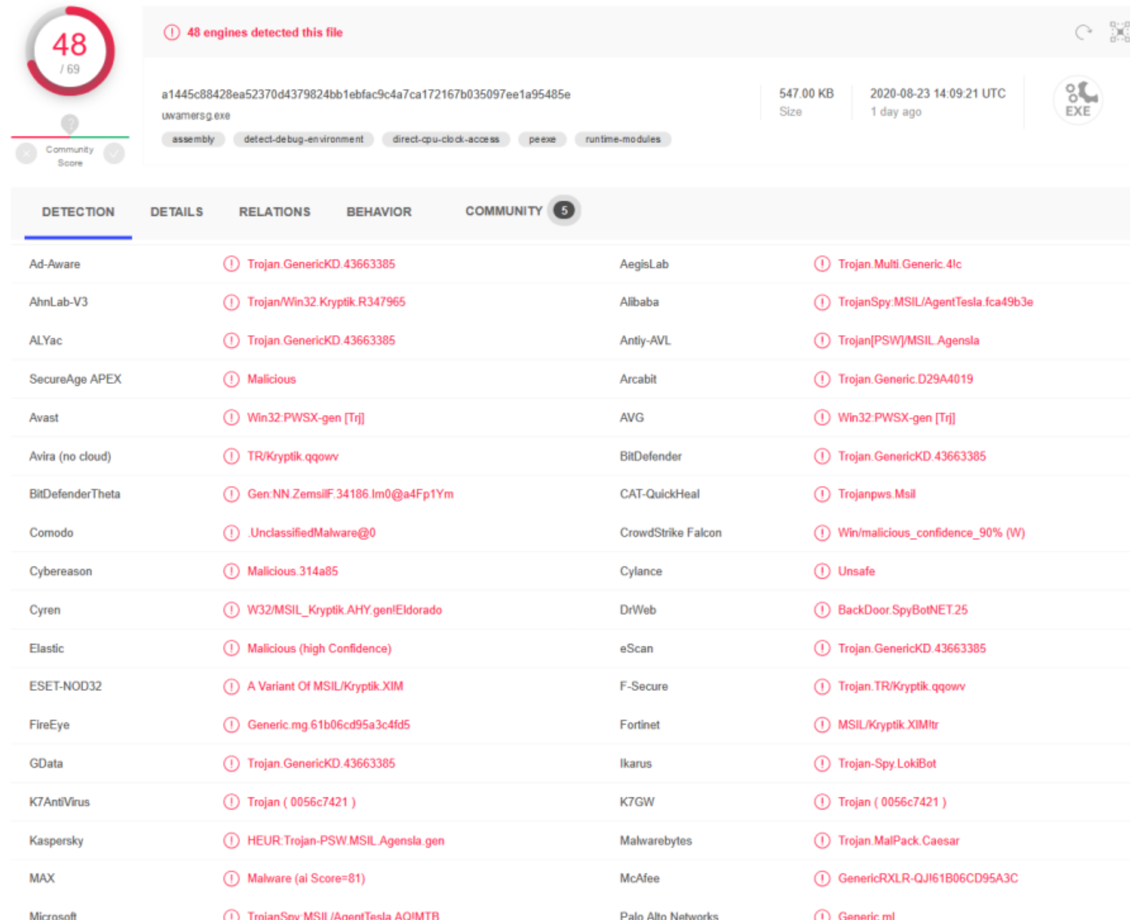


Figure 12: VirusTotal “AgentTesla” mis-identification.

The different obfuscation techniques we observed outline a pattern common to all second-stage executables of operation Epic Manchego. As can be observed in Figure 13, the second stage will dynamically load a decryption DLL. This DLL component then proceeds to extract additional settings and a third-stage payload before transferring the execution to the final payload, typically an information stealer.

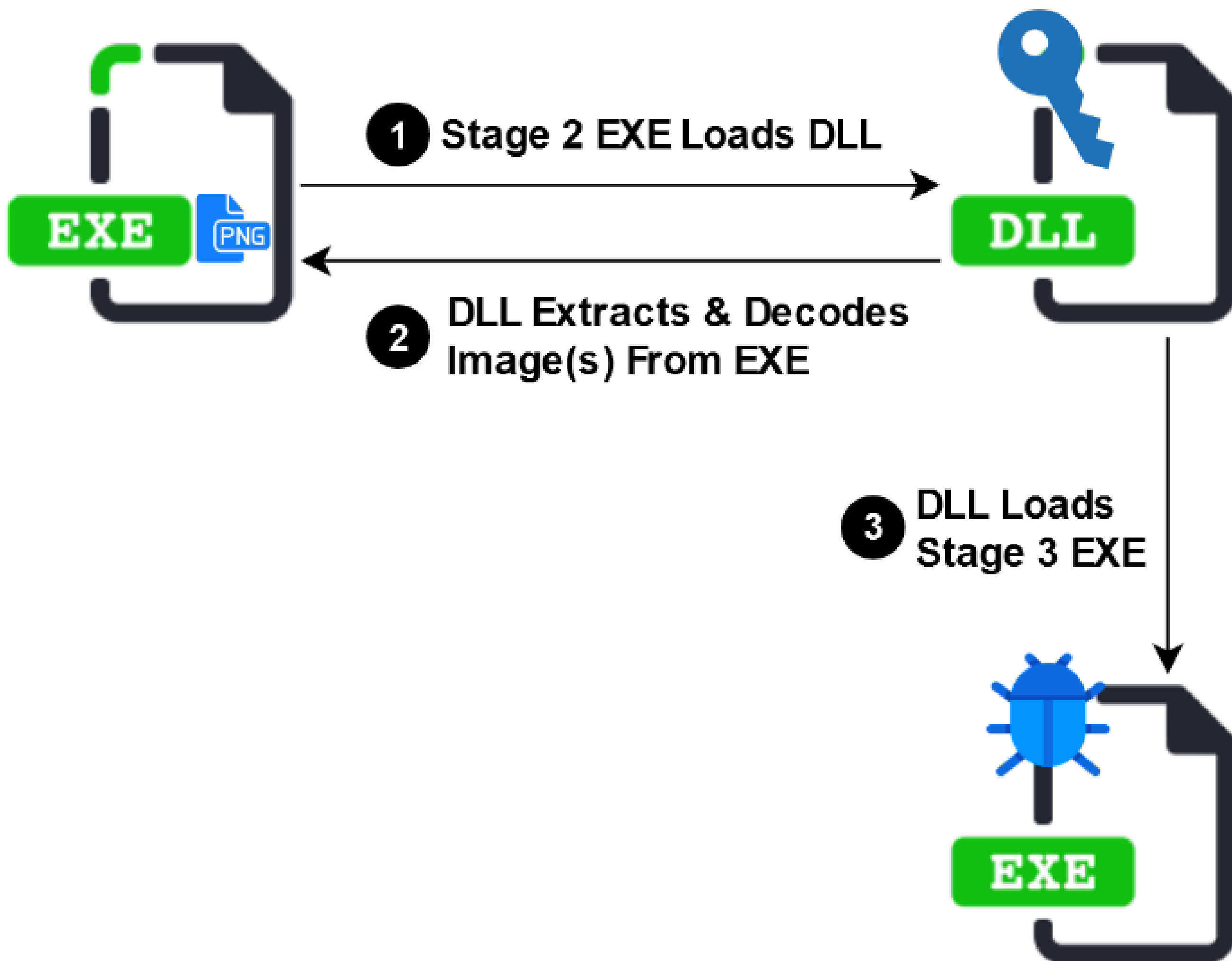


Figure 13: Operation Epic Manchego final stage delivery mechanism.

Although the above obfuscation pattern is common to all samples, we have observed an evolution in its complexity as well as a wide variation in perhaps more opportunistic techniques.

	Early Variants	**Recent Variants**
DLL Component Obfuscation	Obfuscated base64 encoding	Empty fixed-size structures
Final Payload Obfuscation	Single-PNG encoding	Multi-BMP dictionary encoding
Opportunistic Obfuscation	Name randomisation	Run-time method resolving, Goto flow-control, …

Table 1 – Variant comparison

A common factor of the operation’s second-stage samples is the usage of steganography to obfuscate their malicious intent. Figure 14 identifies a partial configuration used in recent variants where a dictionary of settings, including the final payload, is encoded into hundreds of images as part of the second-stage’s embedded resources.

Figure 14: Partial dictionary encoded in a BMP image

The image itself is part of the following second-stage sample which has the following properties:

Filename: crefgyu.exe
MD5: 7D71F885128A27C00C4D72BF488CD7CC
SHA256:
C40FA887BE0159016F3AFD43A3BDEC6D11078E19974B60028B93DEF1C2F95726
File Size: 761 KB (779.776 bytes)
Compilation Timestamp: 2020-03-09 16:39:33

Noteworthy is the likelihood that the obfuscation process is not built by the threat actors themselves. A careful review of the second-stage steganography decoding routine uncovers how most samples mistakenly contain the final payload twice. In the following representation (Figure 15) of the loader’s configuration we can see that its payload is indeed duplicated. The complexity of the second- and third-stage payloads furthermore tend to suggest the operation involves different actors as the initial documents reflect a less experienced actor.

Throughout the multiple dictionary-based variants analyzed we furthermore noticed that, regardless of the final payload, similar keys were used as part of the settings. All dictionaries contained the final payload as “EpkVBztLXeSpKwe” while some, as seen in Figure 15, also contained the same value as “PXcli.0.XdHg”. This suggests a possible builder for payload delivery, which may be used by multiple actors.

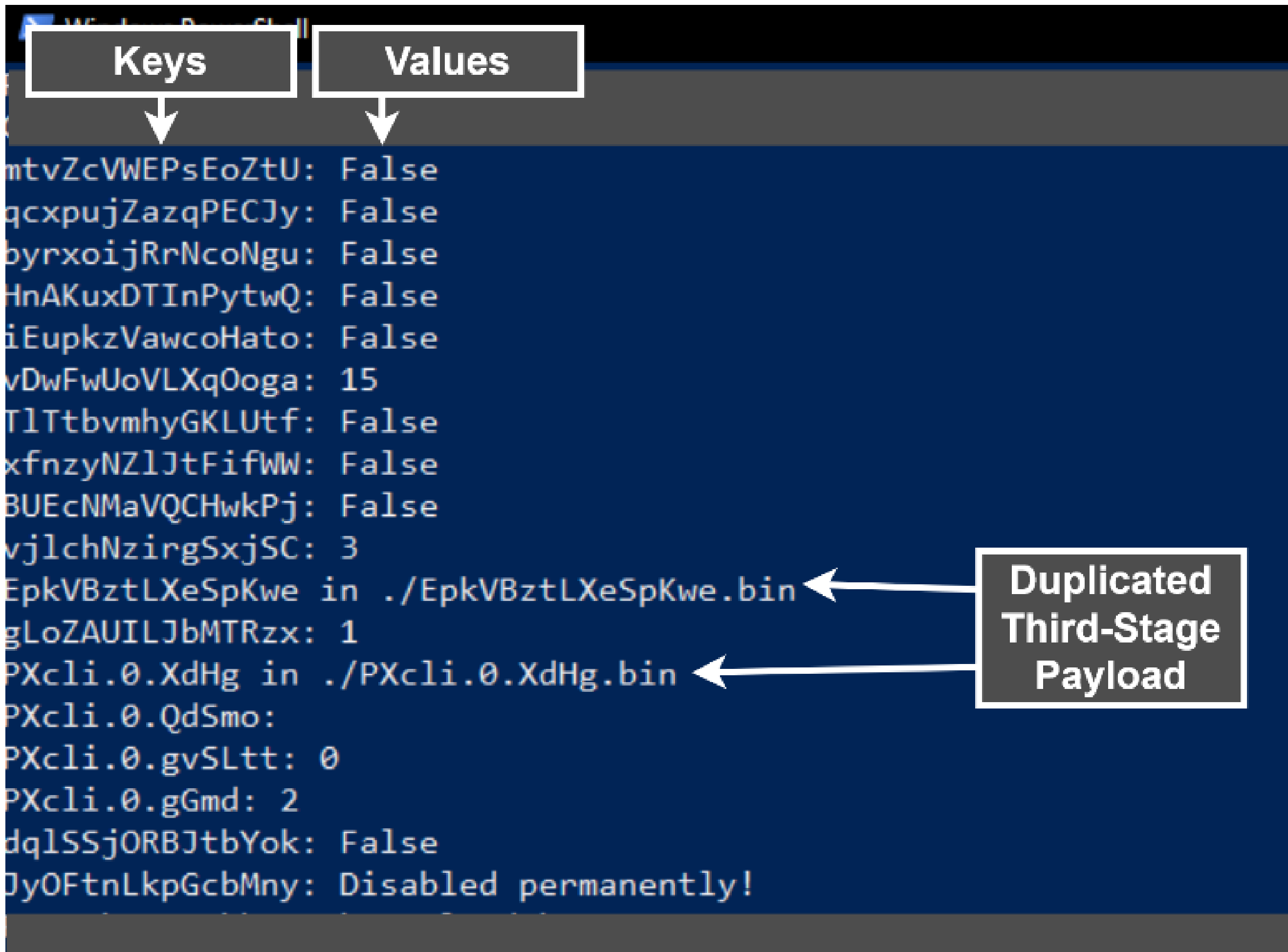


Figure 15: Stage 2 decoded dictionary

Within the manually analyzed dataset of 30 distinct dictionary-based second stages, 19 unique final payloads were observed. From these, the “Azorult” stealer accounts for 50% of the variant’s delivery (Figure 16). Other payloads include “AgentTesla”, “Formbook”, “Matiex” and “njRat”, which are all well-documented already. Both “Azurult” and “njRAT” have a noticeable reusage rate.

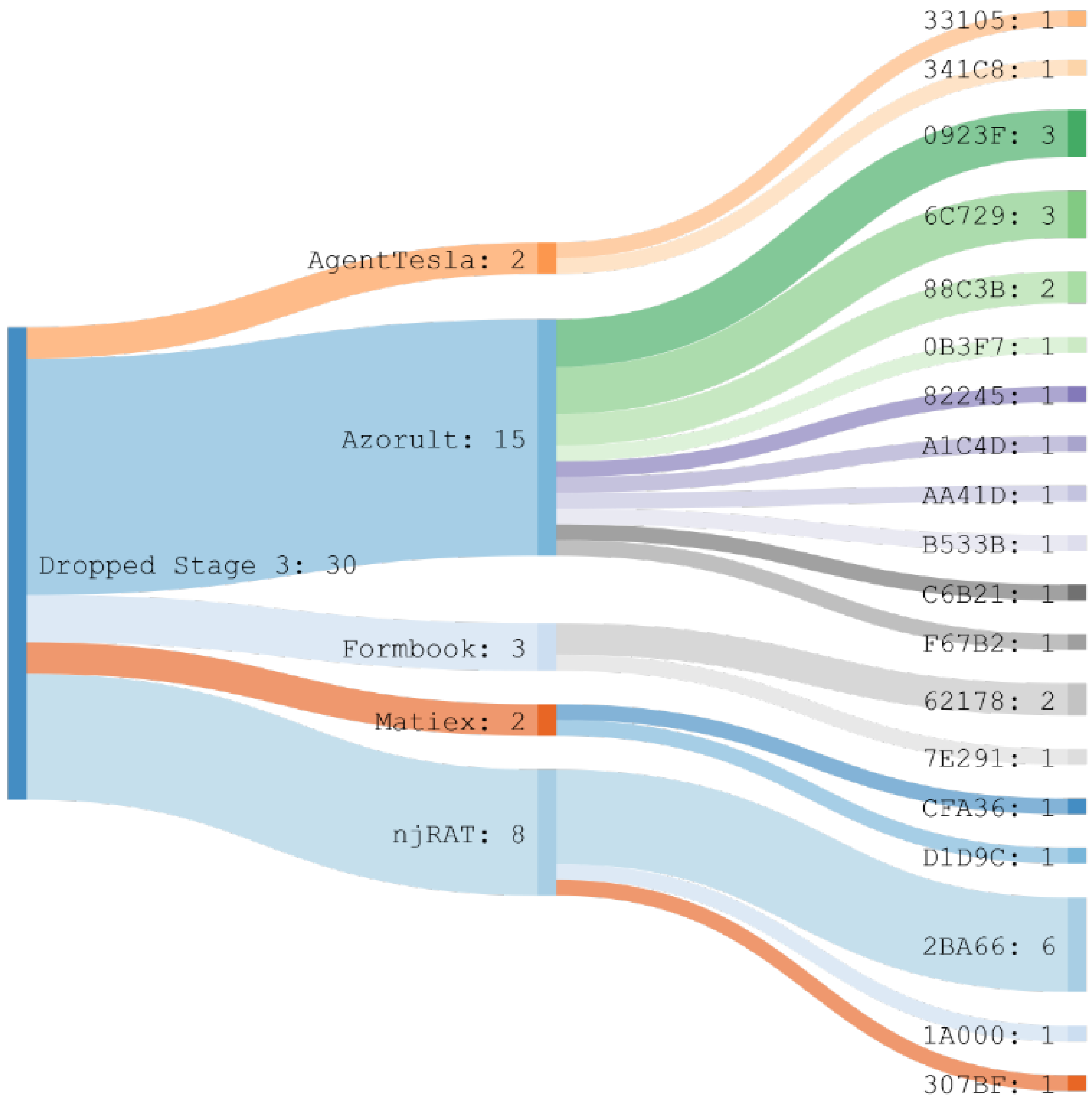


Figure 16: Dictionary-based payload classification and (re-)usage of samples with trimmed hashes

Our analysis of droppers and respective payloads uncovered a common pattern in obfuscation routines. While opportunistic obfuscation methods may evolve, the delivered payloads remain part of a rather limited set of malware families.

Targeting

A small number of the malicious documents we retrieved from VirusTotal were uploaded together with the phishing email itself. Analysis of these emails can shed some light on the potential targets of this actor. Due to the limited number of source emails retrieved, it was not

possible to identify a clear pattern based on the victims. In the 6 emails we were able to retrieve, recipients were in the medical equipment sector, aluminium sector, facility management and a vendor for custom made press machines.

When looking into the sender domains, it appears most emails are sent from legitimate companies. Having used the “Have I Been Pwned”[5] service to confirm if any of the email addresses were known to be compromised, turned up with no results. This leaves us to wonder whether the threat actor was able to leverage these accounts during an earlier infection or whether a different party supplied them. Regardless of who compromised the accounts, it appears the threat actor primarily uses legitimate corporate email accounts to initiate the phishing campaign.

Looking at both sender and recipient, there doesn’t appear to be a pattern we can deduce to identify potential new targets. There does not seem to be a specific sector targeted nor are the sending domains affiliated with each other.

Both body (content) and subject of the emails relate to a more classic phishing scheme, for example, a request to initiate business for which the attachment provides the ‘details’. An overview of subjects observed can be seen below, note some subjects have been altered by the respective mail gateways:

- Re: Quotation required/
- Quote volume and weight for preferred
- *****SPAM***** FW:Offer_10044885_[companyname]_2_09_2020.xlsx*
- [SUSPECTED SPAM] Alternatives for Request*
- Purchase Order Details
- Quotation Request

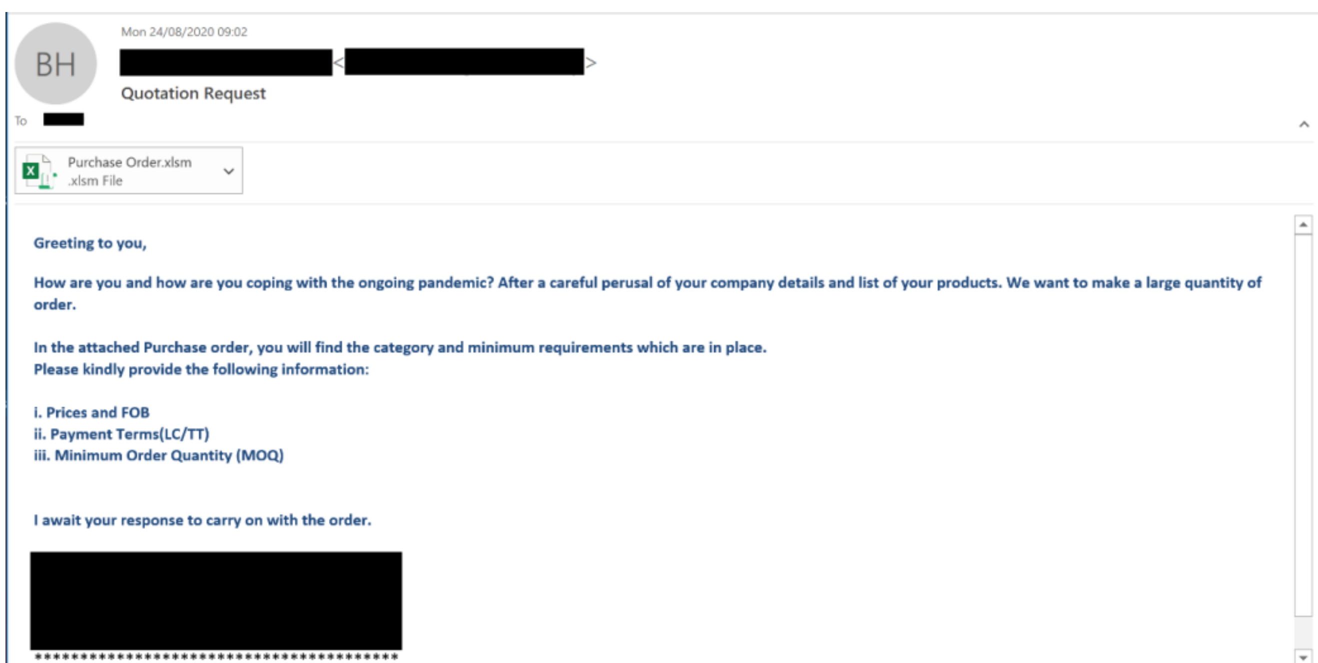


Figure 17 – Sample phishing email

This method of enticing users to open the attachments is nothing new and does not provide a lot of additional information to pinpoint the campaign targeting any specific organisation or verticals.

However, leveraging public submissions of the maldocs through VirusTotal, we clustered over 200 documents, which allowed us to rank 27 countries by submission count without differentiating between uploads possibly performed through VPNs. As shown in Figure 18, areas such as the United States, Czech Republic, France, Germany, as well as China, account for the majority of targeted regions.

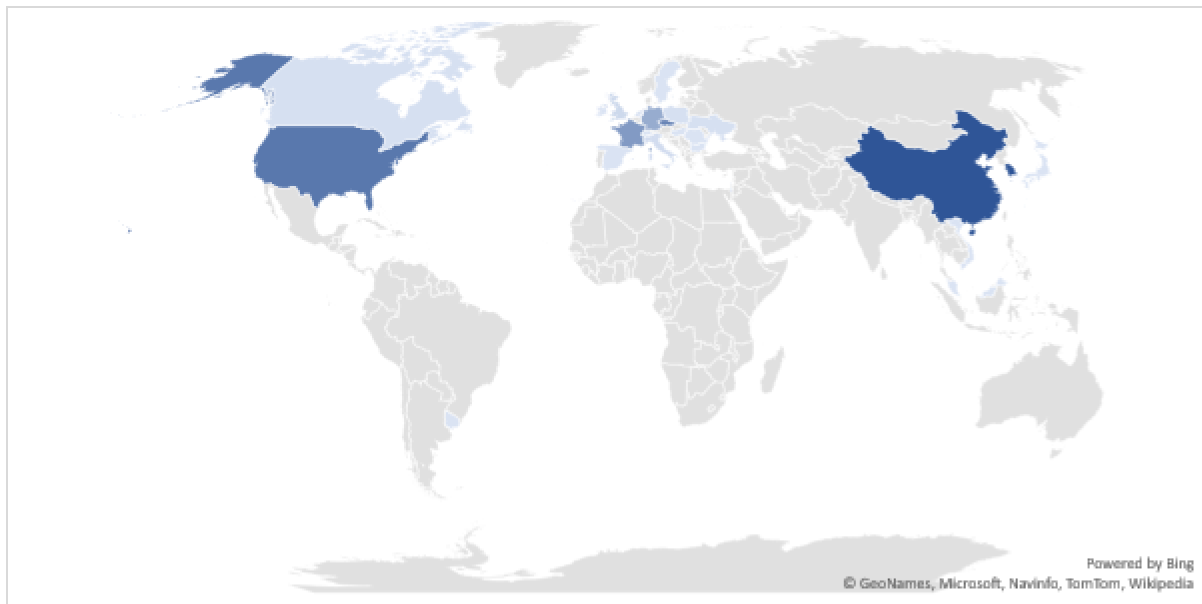


Figure 18 – Geographical distribution of VT submissions

When analysing the initial documents for targeted regions, we primarily identified English, Spanish, Chinese and Turkish language-based images.

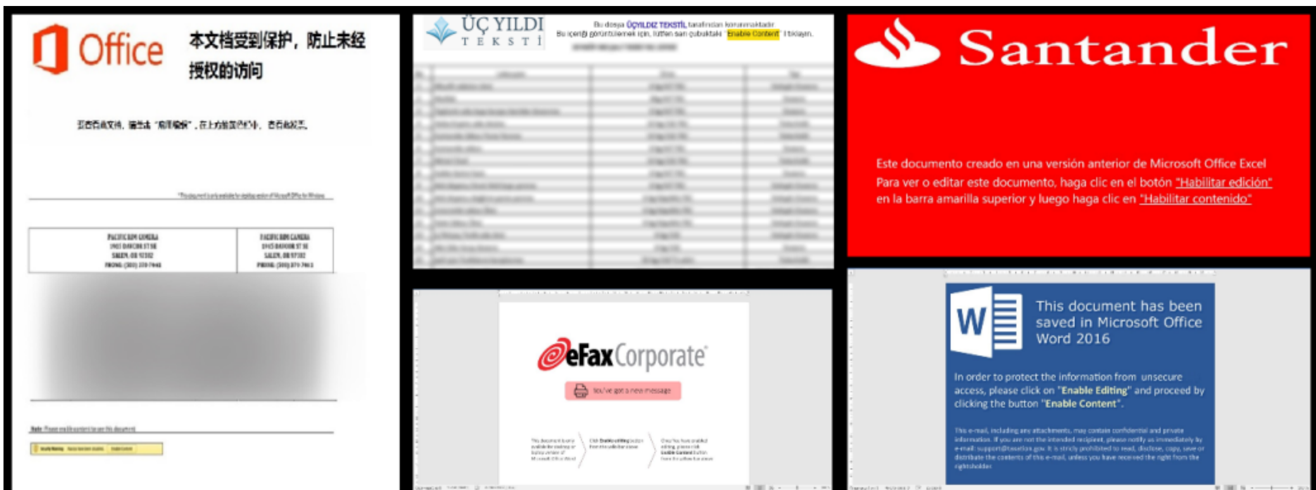


Figure 19 – Maldoc content in Chinese, Turkish, Spanish and English respectively

Some images however contained an interesting detail: some of the document properties are in Cyrillic, and this regardless of the image's primary language.

Although the Cyrillic Word settings were observed in multiple images, a new maldoc detected at time of writing this blog post piqued our interest, as it appears to be the first one to explicitly impersonate a healthcare sector member ("Ohiohealth Hardin Memorial Hospital"), as can be observed in Figure 20. Note also the settings as described above: *СТРАНИЦА 1 ИЗ 1*; which means page 1 of 1.

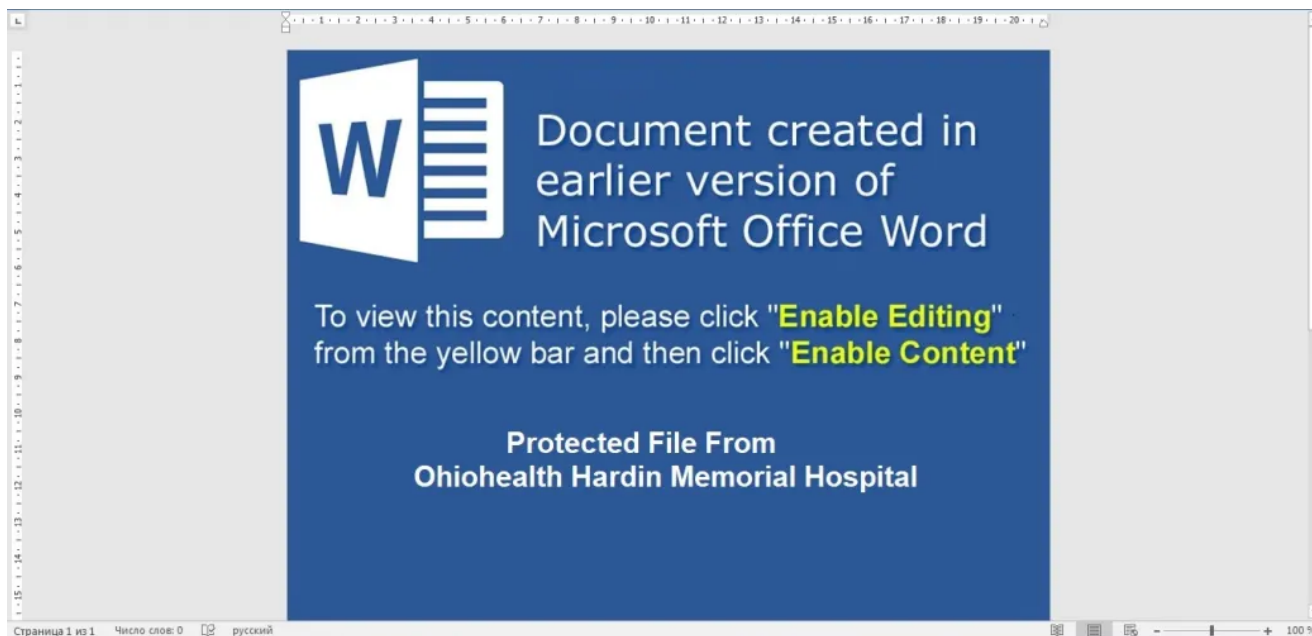


Figure 20 – Maldoc content impersonating "Ohiohealth Hardin Memorial Hospital" with Cyrillic Word settings

This Microsoft Excel document has the following details:

Filename: 새로운 주문 _2608.xlsm (Korean: New order _2608.xlsm)
MD5: 551b5dd7aff4ee07f98d11aac910e174
SHA256: 45cab564386a568a4569d66f6781c6d0b06a9561ae4ac362f0e76a8abfede7bb
File Size: 5.77 KB (5911 bytes)
Earliest Contents Modification: 2020-06-22 14:01:46

While the template from said hospital may have been simply discovered on the web and consequently used by the threat actor, this surprising change in modus operandi does appear to align with the actor's constant evolution observed since the start of tracking.

Assessment

Based on the analysis, NVISO assesses the following:

- The threat actor observed has worked out a new method to create malicious Office documents with a way to at least slightly reduce detection mechanisms;
- The actor is likely experimenting and evolving its methodology in which malicious Office documents are created, potentially automating the workflow;
- While the targeting seems rather limited for now, it’s possible these first runs were intended for testing rather than a full-fledged campaign;
- Recent uptick in detections submitted to VirusTotal confirms the actor may be ramping up their operations;
- While the approach to create malicious documents is unique, the methodologies for payload delivery as well as actual payloads are not, and should be stopped or detected by modern technologies;
- Of interest is a recent blog post published by Xavier Mertens on the SANS diary Tracking A Malware Campaign Through VT[6]. It appears another security researcher has also been tracking these documents, however, they have extracted the VBA code from the maldocs and uploaded that portion. These templates relate to the PowerShell way of downloading the next stage.

In conclusion, NVISO assesses this specific malicious Excel document creation technique is likely to be observed more in the wild, albeit missed by email gateways or analysts, as payload analysis is often considered more interesting. However, blocking and detection of these types of novelties, such as the maldoc creation described in this blog, enables organizations to detect and respond quicker in case an uptick or similar campaign occurs. The recommendations section provides ruling and indicators as a means of detection.

Recommendations

- Filter email attachments and emails sent from outside your organization;
- Implement robust endpoint detect and respond defenses;
- Create phishing awareness trainings and perform a phishing exercise.

YARA

We provide the following rule to implement in your detection mechanisms for use in further hunting missions.

```
rule xlsm_without_metadata_and_with_date {
    meta:
        description = "Identifies .xlsm files created with EPPlus"
        author = "NVISO (Didier Stevens)"
        date = "2020-07-12"
        reference = "http://blog.nviso.eu/2020/09/01/epic-manchego-atypical-maldoc-
delivery-brings-flurry-of-infostealers"
        tlp = "White"
    strings:
        $opc = "[Content_Types].xml"
        $ooxml = "xl/workbook.xml"
        $vba = "xl/vbaProject.bin"
        $meta1 = "docProps/core.xml"
        $meta2 = "docProps/app.xml"
        $timestamp = {50 4B 03 04 ?? ?? ?? ?? ?? ?? 00 00 21 00}
    condition:
        uint32be(0) == 0x504B0304 and ($opc and $ooxml and $vba)
        and not (any of ($meta*) and $timestamp)
}
```

This rule will match documents with VBA code created with EPPlus, even if they are not malicious. We had only a couple of false positives with this rule (documents created with other benign software), and quite some corrupt samples (incomplete ZIP files).

INDICATORS OF COMPROMISE (IOCs)

Indicators of compromise can be found on our Github page here.

MITRE ATT&CK MAPPING

- **Initial Access:**
  T1566.001 Phishing: Spearphishing Attachment
- **Execution:**
  T1204.002 User Execution: Malicious File
- **Defense Evasion:**
  - T1140 Deobfuscate/Decode Files or Information
  - T1036.005 Masquerading: Match Legitimate Name or Location
  - T1027.001 Obfuscate Files or Information: Binary Padding
  - T1027.002 Obfuscate Files or Information: Software Packing
  - T1027.003 Obfuscate Files or Information: Steganography
  - T1055.001 Process Injection: DLL Injection
  - T1055.002 Process Injection: PE Injection
  - T1497.001 Virtualization/Sandbox Evasion: System Checks

Authors

This blog post was created based on the collaborative effort of :