

# Document Analysis and Steganography

Dr. Darryl J D'Souza  
Mob No: 9986382162



# Document Analysis



Documents are a common way of sending or storing information like messages, reports, videos, or ideas. MS Office documents, Images, and audio files are some commonly used formats in our day-to-day lives. However, beyond what we see written in a word document or hear in an audio file, these documents can also contain hidden messages or malicious code that may execute when we open them.

Microsoft Word performs file data validation before opening a file. Data validation is performed in the form of data structure identification, against the OfficeOpenXML standard. If any error occurs during the data structure identification, the file being analysed will not be opened. Usually, Word files containing macros use the .docm extension.



## Microsoft Office Documents

There are two main file formats used by Microsoft Office documents:

- OLE (Object Linking and Embedding)
- OOXML (Office Open XML)

## OLE (Object Linking and Embedding)

### OLE

OLE (Object Linking and Embedding) was the file format used in early versions of Microsoft Office between 1997 and 2003. It defined a “file within a file” structure which allowed other files to be embedded into a file. For example, an Excel spreadsheet could be embedded within a Microsoft Word document.

It supported file extensions like .rtf, .doc, .ppt, and .xls, among others.

## OOXML (Office Open XML)

### OOXML

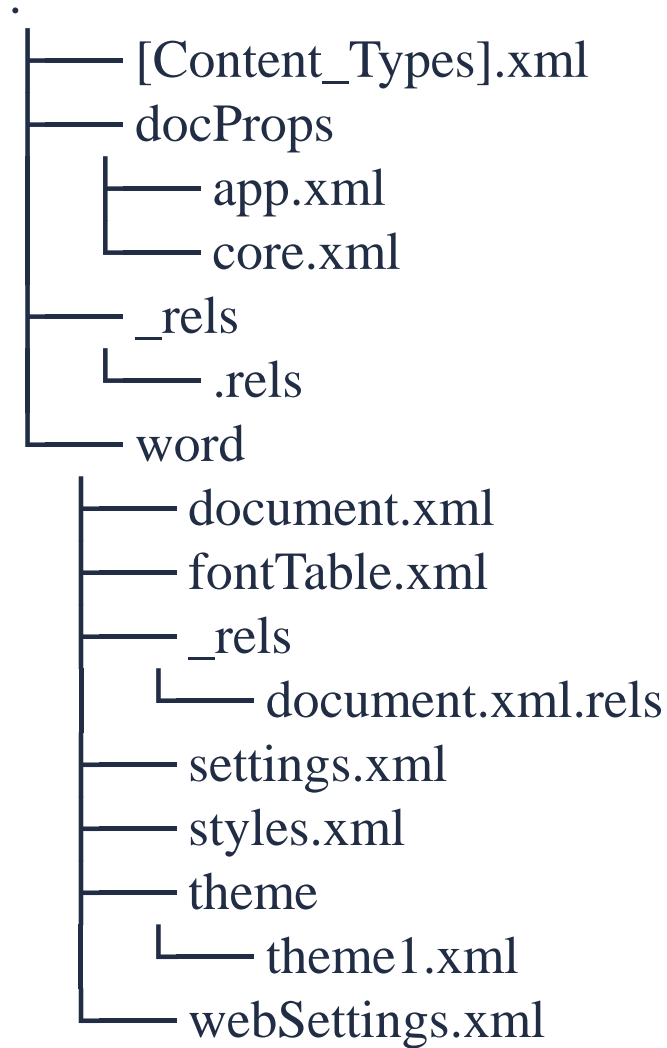
OOXML (Office Open XML) is the current file format used in Microsoft Office, which relies on an XML-based format for office documents.

The extensions for these documents include .docx, .pptx, and .xlsx, among others.

The OOXML format stores Office documents as ZIP containers. This means that the documents such as Word, Excel, and PowerPoint files, are actually just ZIP files. By renaming the extension from .docx, .xlsx, or .pptx to .zip, you can extract the contents of the archive and view the individual XML files. This is a useful feature for digital forensics, as it allows investigators to examine the contents of a document without modifying it.

## Anatomy of an OOXML document

Here's how the file structure of a Word document looks like:



## [Content\_Types].xml

This file contains information about the content types that are present in the document and their corresponding file extensions. This folder contains two files, app.xml and core.xml.

**app.xml** — contains information about the application that was used to create the document.

**core.xml** — contains metadata of the document, such as the author's name, creation date, and modification date.

## \_rels



\_rels

This folder contains one file named .rels.

.rels — contains information about the relationships between the different parts of the document such as for app.xml and core.xml.

## word

This folder contains the actual content of the document.  
document.xml — contains the actual text of the document.

- **fontTable.xml** — contains information about the fonts used in the document.
-  **\_rels** — contains one file document.xml.rels.
  - **document.xml.rels** — contains information about the relationships between the different parts of the document, such as styles, themes, settings, as well as the URIs for external links.
- **settings.xml** — contains document settings and configuration information.
- **styles.xml** — contains information about the styles used in the document.
-  **theme** — contains files about the theme used in the document.
  - theme1.xml — contains the actual theme content.
- **webSettings.xml** — contains information about the web-specific settings of the document, such as HTML frameset settings as well as how the document is handled when saved as HTML.

*The information about any additional files that may be present in a document can be found on the link <http://officeopenxml.com/anatomyofOOXML.php>.*



# Macro-Enabled Documents

---



# Macro-Enabled Documents



An incident responder or forensic investigator should be prepared to examine potentially-malicious document files, which may be located on the compromised system or discovered in email, web, or other network streams. After all, embedding malicious code into documents, such as Excel spreadsheets or Adobe Acrobat PDF files is quite effective at bypassing perimeter defenses.

# Macro-Enabled Documents

Macro-Enabled documents are documents that contain macros, which are sets of instructions that automate tasks. Macros can be written in Visual Basic for Applications (VBA) and can be used to perform a wide range of tasks, such as formatting text, performing calculations, and automating complex processes. However, attackers often utilize this functionality of Office documents with a phishing attack and embed malicious macros to perform malicious actions and install malware on the system.

The extensions for these documents include .docm, .pptm, and .xlsm, among others.

However, it's possible to rename the file by changing the file extension and still keep their macro executing capabilities. For example, an RTF file does not support macros, by design, but a DOCM file renamed to RTF will be handled by Microsoft Word and will be capable of macro execution. The same internals and mechanisms apply to all software of the Microsoft Office Suite (Excel, PowerPoint etc.).

## Why It's a Problem?



As with any program allowing the execution of customizable scripts in the background, attackers can exploit Office suites to run malicious code and compromise victims. Usually the macro-malware acts as a loader in the infection chain, and will download and execute another payload before terminating.

The malware will be embedded in – you guessed it – an Office file, and implanted somewhere for the victim to access it, say a common file share or by e-mail. Once the file is opened, the malware will be executed. This is called a Spearphishing Macro Attack (MITRE T1193: Spearphishing Attachment[1]), and it's been prevalent for a very long time.

## Macro Case study



One memorable example of a macro-malware is the Melissa virus, which first appeared in 1999. The malware was embedded in an MS Word file and when opened, would e-mail itself to the first 50 contacts in the victim's address book. Though it didn't cause much damage to individual users (aside from accidentally causing several mail services to DDOS), the total worth of damages caused was estimated at 80\$ million, and the attacker was sentenced to 20 months in federal prison.

But that is not to say that spearphishing macro attacks are a remnant of days past. According to data gathered by Cofense Intelligence[2], during the month of August 2018, malicious macros embedded in MS Office files accounted for 45% of all malware delivery mechanisms analyzed. Moreover, despite the seemingly amateur nature of such attacks, the malwares were used to deliver some of the most malignant payloads in circulation – including Geodo, Chanitor, AZORult and others.

## User Execution

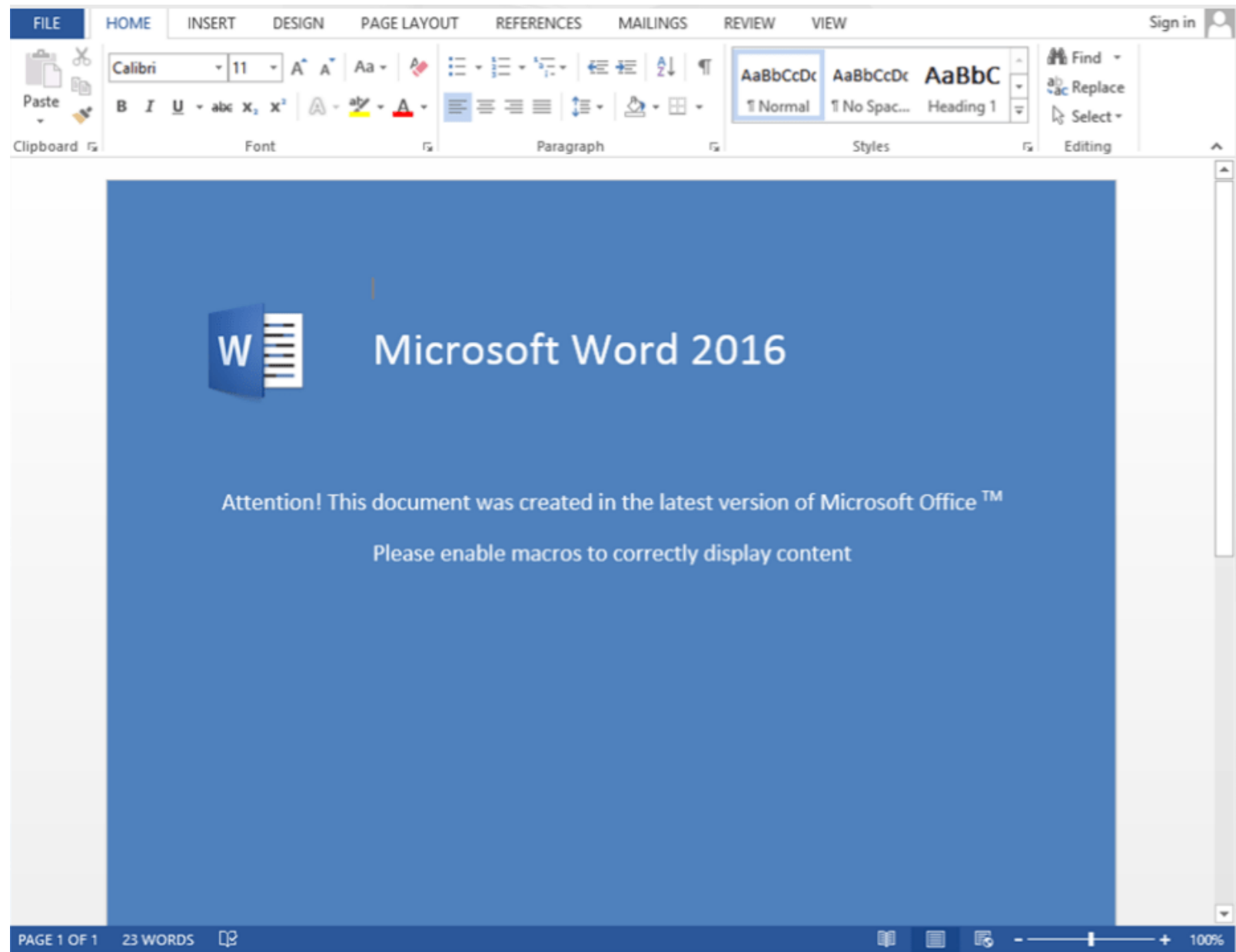


In today's ever-evolving defense landscape, spearphishing attacks rely on a unique vulnerability, and one that is very hard to mitigate completely – human error. The adversary waits for certain actions to be taken by the victim (say, opening a file) to gain execution, rather than on vulnerabilities that are dependent on the victim's machine alone .

Furthermore, MS Office macro-execution is usually either enabled by default or is allowed by a single mouse click upon opening the malicious file. This creates a large margin for user error and increases the probability of a successful attack.

Relying on user execution also allows adversaries to target and lure non-technical users specifically, compromising entire enterprises through them. But mitigation is still possible, as we'll see below, by implementing proper endpoint defense mechanisms as well as educating users.

# MS Office macro lures

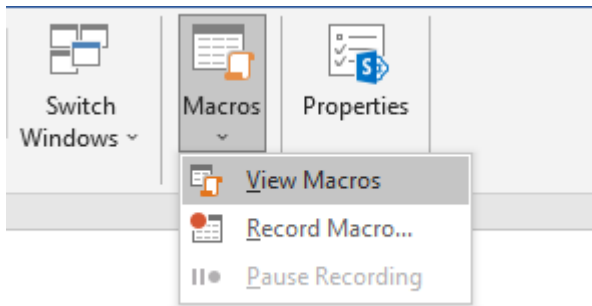


As gathered by Microsoft's Threat Intelligence Center over the summer of 2016

## *For You to try:*

Create an empty word document, and follow the steps below:

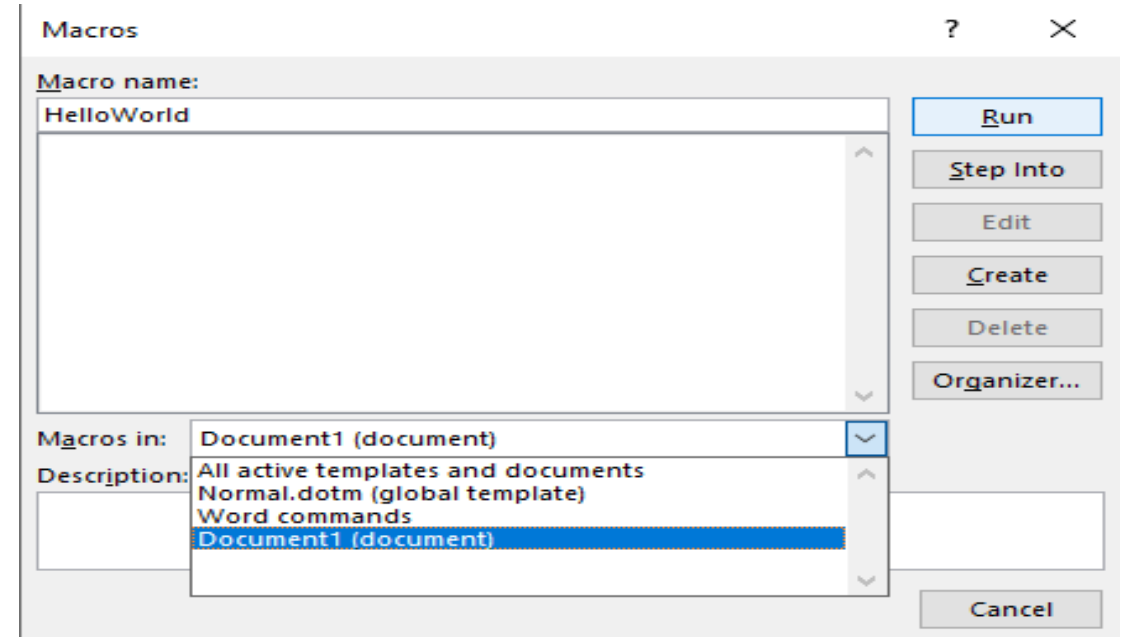
1. Click View → Macros → View Macros.



3. Paste the following code in the text box.

```
Sub HelloWorld()  
Dim doc As Document  
Set doc = Word.ActiveDocument  
doc.Content.InsertAfter ("Hello, World!")  
End Sub
```

2. Type a name such as HelloWorld, select Document1 (current document) under Macros in, and click create.



4. Close the Microsoft Visual Basic for Application tab.
5. Repeat step 1, select the HelloWorld macro and click Run.
6. Observe that Hello, World! is now written in the document.
7. Save the document as .docm.

## Tools to analyze malicious documents

As per the anatomy of OOXML files, the macro is now stored inside word/vbaProject.bin, however, we won't be able to read it as it's in binary form. But, we can use a collection of tools called oletools to analyze and extract macros from OLE files such as Microsoft Office Documents.

oletools is a package of python tools to analyze Microsoft OLE2 files (also called Structured Storage, Compound File Binary Format or Compound Document File Format), such as Microsoft Office documents or Outlook messages, mainly for malware analysis, forensics and debugging. It is based on the olefile parser.

The recommended way to download and install/update the latest stable release of oletools is to use pip:

To get the latest development version instead:

On Linux/Mac: `sudo -H pip install -U`

`https://github.com/decalage2/oletools/archive/master.zip`

On Windows: `pip install -U https://github.com/decalage2/oletools/archive/master.zip`



# Tools to analyze malicious documents

## Tools to analyze malicious documents

- [oleid](#): to analyze OLE files to detect specific characteristics usually found in malicious files.
- [olevba](#): to extract and analyze VBA Macro source code from MS Office documents (OLE and OpenXML).
- [MacroRaptor](#): to detect malicious VBA Macros
- [msodde](#): to detect and extract DDE/DDEAUTO links from MS Office documents, RTF and CSV
- [pyxswf](#): to detect, extract and analyze Flash objects (SWF) that may be embedded in files such as MS Office documents (e.g. Word, Excel) and RTF, which is especially useful for malware analysis.
- [oleobj](#): to extract embedded objects from OLE files.
- [rtfobj](#): to extract embedded objects from RTF files.

## Tools to analyze the structure of OLE files

- [olebrowse](#): A simple GUI to browse OLE files (e.g. MS Word, Excel, Powerpoint documents), to view and extract individual data streams.
- [olemeta](#): to extract all standard properties (metadata) from OLE files.
- [oletimes](#): to extract creation and modification timestamps of all streams and storages.
- [oledir](#): to display all the directory entries of an OLE file, including free and orphaned entries.
- [olemap](#): to display a map of all the sectors in an OLE file.

## Example: Malicious VBA Macro (Ping Example)

The following VBA (Visual Basic for Applications) macro, if embedded in a Word/Excel document, executes a simple **ping** command when macros are enabled.

### Code:

```
Sub AutoOpen()  
    Call RunPayload  
End Sub
```

```
Sub RunPayload()  
    Dim cmd As String  
    cmd = "cmd /c ping 8.8.8.8"  
    Shell cmd, vbHide  
End Sub
```

### Explanation:

**1.AutoOpen():** Runs automatically when the document is opened if macros are enabled.

**2.RunPayload():** Executes the Windows command prompt (cmd) to send a ping request to Google's DNS server (8.8.8.8).

**3.Shell cmd, vbHide:** Runs the command in hidden mode to avoid raising suspicion.

## Example: Malicious VBA Macro (Ping Example)



### Malicious Adaptations:

- Instead of ping, an attacker could use powershell or certutil to download malware.
- Modify AutoOpen to trigger payloads during events like document closing (AutoClose).
- Create registry persistence to execute the payload even after the document is closed.

### Forensic Detection:

- **Analyze VBA Code:** Use oledump.py to extract and inspect macros.
- **Check System Logs:** Look for unusual process executions (e.g., cmd.exe spawning from WINWORD.EXE).
- **Network Traffic Monitoring:** Detect suspicious pings or connections

# Checking System Logs for Unusual Process Executions

When investigating malicious macros, forensic analysts check system logs to find suspicious process executions. If a macro executes a command like `cmd.exe` or `powershell.exe`, it may be logged in Windows Event Viewer or captured using Sysmon.

## Using Windows Event Viewer

Windows logs process creation events under **Event ID 4688 (Process Creation)** in the **Security** log.

### Steps to Detect `cmd.exe` from `WINWORD.EXE`

#### 1. Open Event Viewer:

- Press Win + R, type `eventvwr.msc`, and press Enter.

#### 2. Navigate to Security Logs:

- Go to Windows Logs → Security.

#### 3. Filter for Process Creation (Event ID 4688):

- Click "**Filter Current Log**" (right panel).
- In the **Event ID** field, enter: 4688
- Click **OK**.

#### 4. Analyze Process Parent-Child Relationship:

- Look for entries where:
  - New Process Name = `cmd.exe`
  - Creator Process Name = `WINWORD.EXE` (or `EXCEL.EXE`)
- If found, it indicates that a Word document triggered `cmd.exe`.

# Checking System Logs for Unusual Process Executions

When investigating malicious macros, forensic analysts check system logs to find suspicious process executions. If a macro executes a command like cmd.exe or powershell.exe, it may be logged in Windows Event Viewer or captured using Sysmon.

## Using Sysmon (More Detailed Logging)

Sysmon (System Monitor) from **Microsoft Sysinternals** provides more advanced process tracking.

### Steps to Set Up and Detect Malicious Macros

#### 1. Install Sysmon:

- Download from [Microsoft Sysinternals](#).
- Run:

```
cmd  
sysmon -accepteula -i
```

#### 2. Check for Suspicious Process Execution (Sysmon Event ID 1)

- Open **Event Viewer** → Applications and Services Logs → Microsoft → Windows → Sysmon → Operational.
- Look for **Event ID 1** (Process Create).
- Check ParentImage = WINWORD.EXE
- Check Image = cmd.exe or powershell.exe.

*For You to try: Try both the examples mentioned*




- 1. Use oleid to detect whether our document has any macros embedded in it.**
- 2. Use olevba to extract the macros from the document.**



# Steganography

---



The word steganography comes from the Greek word Steganographia, made up of two words “steganos” meaning “covered or concealed” and “graphia” meaning “to write”. It involves hiding secrets in an otherwise seemingly innocent piece of information. An early example of steganography is the use of invisible ink made from lemon juice or vinegar to write on paper and then reveal the writing by heating the paper.

## Steganography



In today’s digital world, steganography is used to hide a message within another file, like an image or audio file, in such a way that it can not be seen or heard by anyone who doesn't know it's there. This technique can be used for both innocent purposes, like sending a secret message to a friend, or for malicious reasons, like concealing evidence or communicating without detection.

As a form of covert communication, steganography is sometimes compared to cryptography. However, the two are not the same since steganography does not involve scrambling data upon sending or using a key to decode it upon receipt.



The background features several large, overlapping geometric shapes, primarily diamonds and triangles, in teal, yellow, and green colors. These shapes are arranged in a way that creates a sense of depth and movement, with some shapes appearing to be layered on top of others. The overall aesthetic is modern and abstract.

“

Today, digital steganography is one of the important components in the toolboxes of spies and malicious hackers, as well as human rights activists and political dissidents.

— Ben Dickson

“

The first recorded uses of steganography can be traced back to 440 BC in Greece, when Herodotus mentions two examples in his Histories. Histiaeus sent a message to his vassal, Aristagoras, by shaving the head of his most trusted servant, "marking" the message onto his scalp, then sending him on his way once his hair had regrown, with the instruction, "When thou art come to Miletus, bid Aristagoras shave thy head, and look thereon."



## How steganography works

Steganography works by concealing information in a way that avoids suspicion. One of the most prevalent techniques is called ‘least significant bit’ (LSB) steganography. This involves embedding the secret information in the least significant bits of a media file. For example:

- In an image file, each pixel is made up of three bytes of data corresponding to the colors red, green, and blue. Some image formats allocate an additional fourth byte to transparency, or ‘alpha’.
- LSB steganography alters the last bit of each of those bytes to hide one bit of data. So, to hide one megabyte of data using this method, you would need an eight-megabyte image file.
- Modifying the last bit of the pixel value doesn’t result in a visually perceptible change to the picture, which means that anyone viewing the original and the steganographically-modified images won’t be able to tell the difference.

The same method can be applied to other digital media, such as audio and video, where data is hidden in parts of the file that result in the least change to the audible or visual output.

Other steganography methods include hiding an entire partition on a hard drive or embedding data in the header section of files and network packets. The effectiveness of these methods depends on how much data they can hide and how easy they are to detect.

## How is this different from cryptography?

It may immediately occur to us that this is similar to cryptography, but it is not so. In cryptography, the objective is to modify the original message in such a fashion it becomes difficult to get to the original message from the modified message. The original and modified messages are explicitly expected to look different. While in image steganography, the objective is to deceptively hide a message within another original message and thereby, modifying it. The modified message is expected to look very similar to the original message.

Let's consider a scenario where two employees are expected to exchange official messages via email. In this case, we know that they will exchange messages, and hence, there is no need for them to hide those messages. However, as those messages might contain sensitive information, they might want to encrypt-decrypt it using cryptography. On the contrary, consider information exchange between a spy, operating undercover in the opponents' military regiment and his parent organization, with whom he wants to share important information.

In this case, the officers would be monitoring pretty much everything that goes in and out of his room. Hence, if he resorts to using cryptography to share encrypted information, officers might get suspicious. In such a situation, he might prefer image steganography, where he would deceptively hide a message within another object/message, without raising suspicions, and find a way to drop it off to his recipient. In summary, cryptography hides the meaning of the data, while steganography hides the existence of the data. Although they are different techniques, they might as well be used in combination, in the same instance, to get the best of both worlds.

## Use cases or applications of steganography

Although the prime objective is to share messages or information discreetly, it has found varied fields of applications such as

- Hackers using steganography techniques for malware transmission
- Intelligence agencies use them for communication.
- Printers also use micro-dots as a steganography tool to embed timestamps and date information within the document. Also, the same technique is used in bank-note printing, to prevent colour copiers from reproducing images of currency as fake-notes.

## Case Study



Some recent examples include:

- Malicious memes on Twitter — [Link to blog](#).
- PDFs with malicious JavaScript code — [Link to article](#).

# Image Steganography

As the name suggests, Image Steganography refers to the process of hiding data within an image file. The image selected for this purpose is called the cover image and the image obtained after steganography is called the stego image.

PNG and JPEG are two common image formats. They can also be used as a channel to hide messages inside them. Both these formats rely on different structures to construct an image, and therefore, different techniques are utilized to hide messages within them.

Hiding information within each medium or file type involves a different technique and each technique has its own set of pros and cons and is appropriately deployed per requirement. In this article, we will look at the most popular method of hiding information within an image file using a technique called the Least Significant Bit (LSB). We will also look at an implementation of the same

# Understanding Image steganography

Before diving into steganography, it is important to understand pixels and colour models. A pixel is the smallest building of an image and the colours in any pixel are (assume RGB) a function of the combination of proportions of red, green, and blue.

So a pixel with a value of 0, 0, and 1 would mean 0 parts of red, 0 parts of green and 1 part of blue; in essence, this would turn out to be a blue pixel. In the case of an 8-bit system, a pixel can accommodate up to 8 digits (zeros or ones), and the largest number that could be represented in 8 digits is 11111111 which would be 255, and the smallest number that could be represented in 8 digits, would be 00000000 which would be 0. So any pixel in an 8-bit scenario could accommodate anything between 0 to 255 as a value for each of the colours. Now let's say a random 8-bit grid has 3 pixels and each pixel having the below values for R, G, and B.

	The proportion of Red (R)	The proportion of Green (G)	The proportion of Blue (B)
Pixel 1	00101101	00011100	11011100
Pixel 2	10100110	11000100	00001100
Pixel 3	11010010	10101101	01100011



# Understanding image steganography

And if we want to house a secret number 200, we get the binary value of that number, i.e, 11001000. and use each digit of that number to replace the least significant digit (mostly the last digit) of our pixel grid, indicated in bold red font. The new colour scheme would be as below:

	The proportion of Red (R)	The proportion of Green (G)	The proportion of Blue (B)
Pixel 1	0010110 <b>1</b>	0001110 <b>1</b>	1101110 <b>0</b>
Pixel 2	1010011 <b>0</b>	1100010 <b>1</b>	0000110 <b>0</b>
Pixel 3	1101001 <b>0</b>	1010110 <b>0</b>	0110001 <b>1</b>

This would alter colours in the original image in the three channels for the 3 pixels by the smallest amount, thereby rendering the alerted image almost indistinguishable from the original image.

# PNG

For PNG files, one of the commonly employed techniques to hide messages is by modifying the least significant bits (LSBs) of certain pixels such as those who have minimal impact on the quality of the image. The algorithm then knows which pixels to extract the embedded message from.

One tool used for detecting PNG steganography is **zsteg**



*Example image of Van Gogh's Starry Night painting with a message already embedded in it. Download the image from teams Lab -3*



For JPEG files, a commonly used steganography method involves finding pairs of positions inside an image such that exchanging their values has the effect of embedding the corresponding part of the secret message. If no such pairs are found, the pixels in the remaining positions are simply overwritten.



With steghide tool, we can hide and extract messages out of JPEG files.

## JPEG

💡 Another tool to add to your digital forensics toolkit is exiftool, which can be useful in extracting metadata from files such as an image or an audio file, which includes information such as the creation and modification dates, author, and location of where an image was captured (latitude and longitude).

# Lab Questions

1. A phishing attack has been reported in your organization, where an employee received a malicious Word document in an email that appeared to come from a trusted source. The employee opened the document which had macros in it, resulting in the attacker gaining access to the employee's computer. A secret which will reveal the attacker's identity, is embedded inside the macro code. You are tasked with analyzing the macro code and extracting the embedded secret. The secret has the format `flag{s0me_str1ng}`.
  - The Word document – Yearly Bonus
2. A mole within the government has leaked top secret information to a spy. The mole, aware of spycraft techniques, used steganography to hide the information within an image, which he then slipped to his handler. The spy received the image and pasted it into a PowerPoint document, covering it with multiple random images to conceal it. One of our spies has gained access to the enemy spy's computer and recovered the PowerPoint document. Your mission is to extract the first image, extract the top secret information as well as the name and location of his source inside the government.

The PowerPoint document – Presentation.

Tools to explore – stegseek & steghide– Jpeg image

zsteg – png image

exiftool