SIGPwny

General FA2025 • 2025-12-4

Forensics

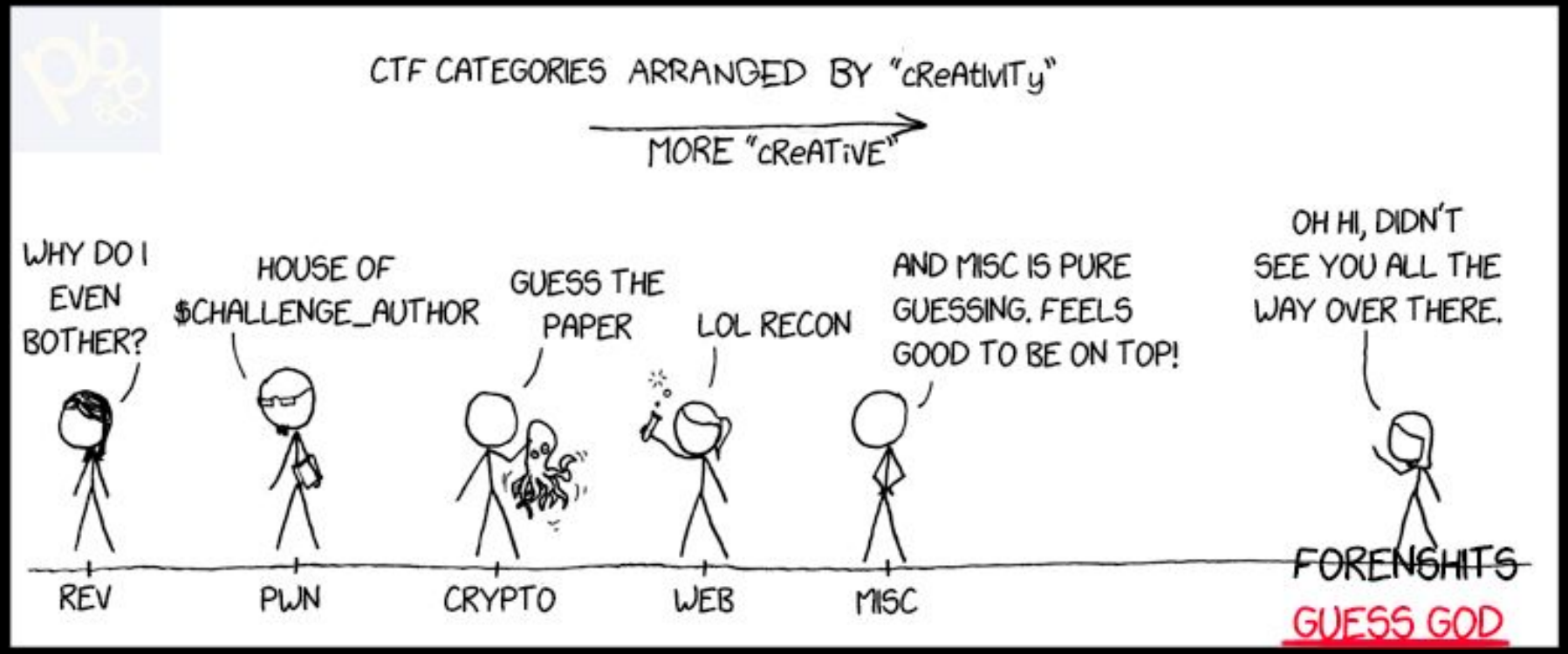
Bryce Kurfman & Suchit Bapatla

Announcements

- Minecraft/Gaming Social
  - Sunday at 5:00 pm
  - Somewhere in Siebel or DCL
- Good luck with finals!

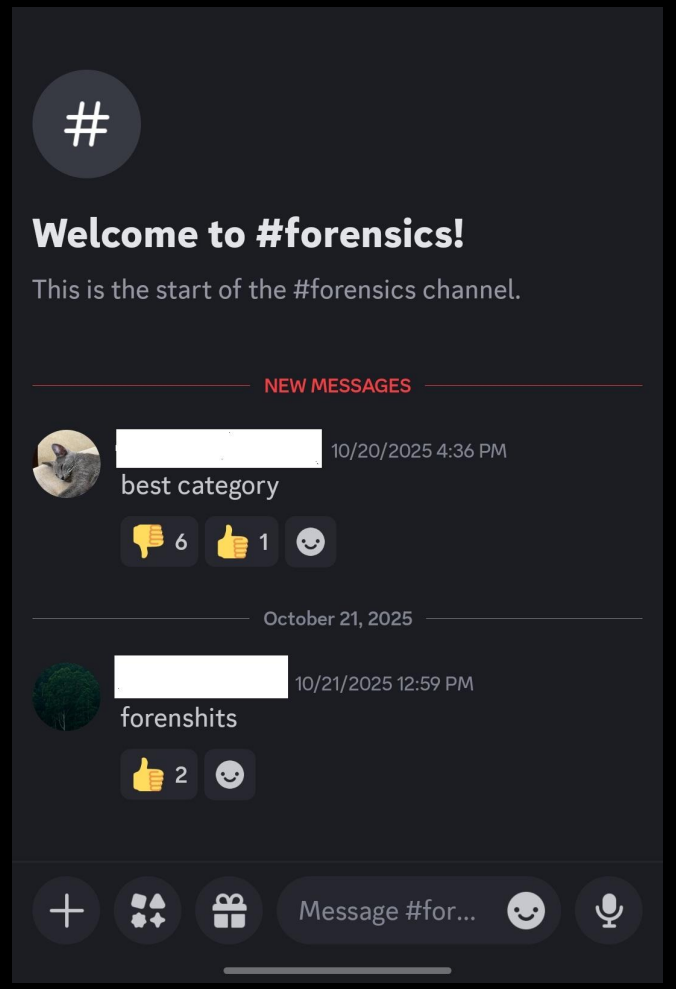
ctf.sigpwny.com

sigpwny{on_the_loose}



What is Digital Forensics?

- File Forensics
  - File format
  - Steganography
- OS Forensics
  - aka Incident Response
- Memory Forensics
  - Dump file of RAM of process or entire OS
- Sleuth Kit



File Forensics

file

File extension is just a *hint* to users and OS about the file's *intended* purpose, but the content can be arbitrary.

`file` command uses "magic bytes" or a file signature to determine the format of a file (e.g. PNG files always start with `89 50 4E 47`)

Q:

```
$ file unknown.txt
unknown.txt: JPEG image data, JFIF standard 1.01, aspect ratio, density 1x1,
 segment length 16, progressive, precision 8, 400x400, components 3
```

binwalk

Finds files formats appended in a file (like `file` but recursive). For example, some binaries include their assets!

```
$ binwalk cmd.exe

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Microsoft executable, portable (PE)
235448        0x397B8         Copyright string: "Copyright (c) Microsoft Corpora
253789        0x3DF5D         mcrypt 2.5 encrypted data, algorithm: "o_exit", ke
283472        0x45350         XML document, version: "1.0"
283534        0x4538E         Copyright string: "Copyright (c) Microsoft Corpora
294568        0x47EA8         PNG image, 256 x 256, 8-bit/color RGBA, non-interl
294609        0x47ED1         Zlib compressed data, default compression
```

Usage:

`binwalk <FILE>` - View a list of contained file formats

`binwalk -e <FILE>` - Extract each file format from the file

binwalk

Sometimes it’s normal to find other file format in one file, as one file format may use another to encode information.

Example: Images in PDF

```
cbcicada@DESKTOP-5T74EEM:/mnt/c/Users/CBCicada/Downloads$ binwalk MP3_FA24_CP2.pdf

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PDF document, version: "1.5"
78            0x4E            Zlib compressed data, best compression
1319          0x527           Zlib compressed data, best compression
3237          0xCA5           Zlib compressed data, best compression
4368          0x1110          Zlib compressed data, best compression
7435          0x1D0B          Zlib compressed data, best compression
10269         0x281D          Zlib compressed data, best compression
11621         0x2D65          Zlib compressed data, best compression
14478         0x388E          Zlib compressed data, best compression
17128         0x42E8          Zlib compressed data, best compression
20555         0x504B          Zlib compressed data, best compression
23832         0x5D18          Zlib compressed data, best compression
25324         0x62EC          Zlib compressed data, best compression
```

strings

Lists plaintext strings that exist in a file, useful for binary files

Usage:

`strings <FILE>` - print all strings of length 4 or greater

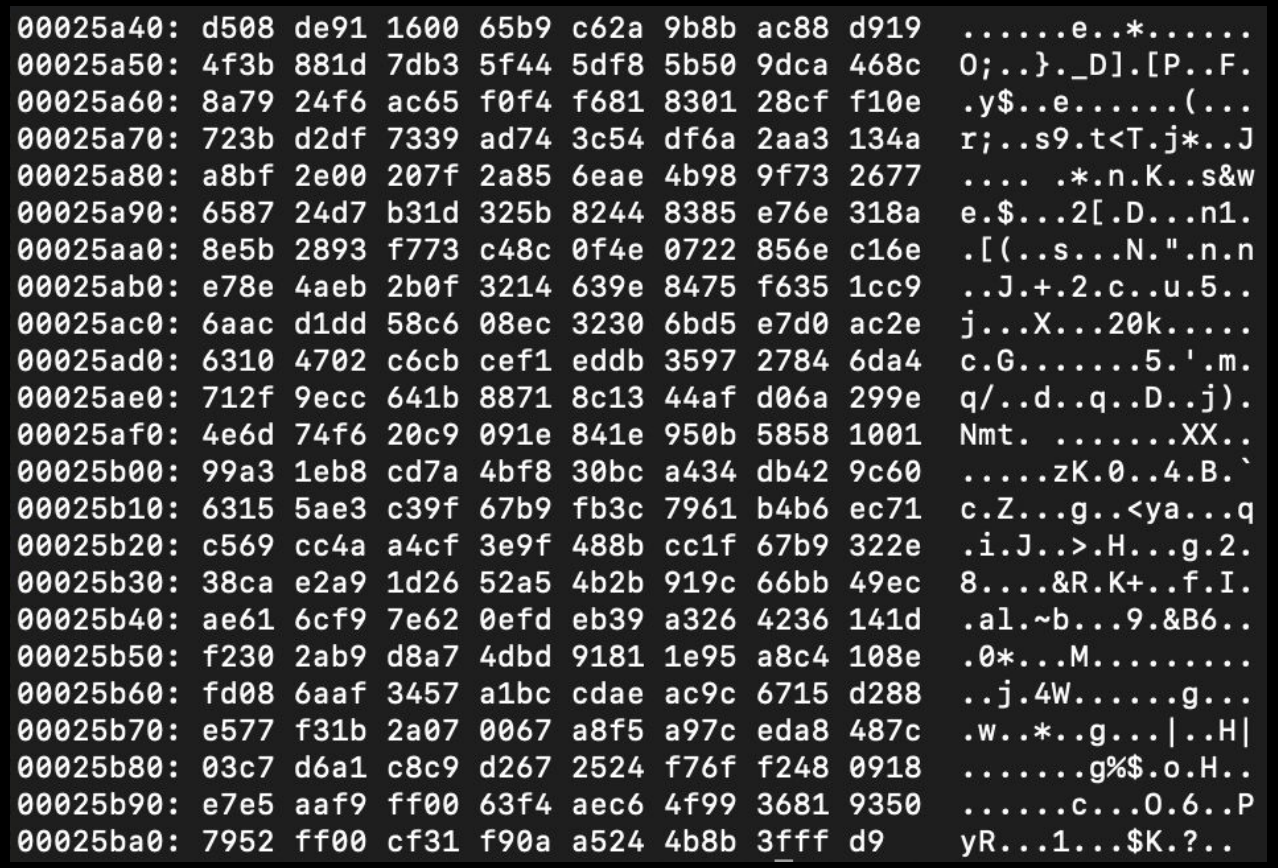
`strings -n 16 <FILE>` - print all strings of length 16 or greater

```
$ strings -n 16 cmd.exe
!This program cannot be run in DOS mode.
SetThreadUILanguage
Unknown exception
bad array new length
api-ms-win-core-winrt-l1-1-0.dll
ext-ms-win-branding-winbrand-l1-1-0.dll
ext-ms-win-cmd-util-l1-1-0.dll
ext-ms-win-appmodel-shellexecute-l1-1-0.dll
onecore\internal\sdk\inc\wil\opensource\wil\resource.h
WilFailureNotifyWatchers
RtlRegisterFeatureConfigurationChangeNotification
RtlUnregisterFeatureConfigurationChangeNotification
RtlNotifyFeatureUsage
NtQueryWnfStateData
NtUpdateWnfStateData
onecore\internal\sdk\inc\wil/Staging.h
CMD Internal Error %s
Null environment
APerformUnaryOperation: '%c'
APerformArithmeticOperation: '%c'
IsDebuggerPresent
SetConsoleInputExeNameW
RaiseFailFastException
RtlNtStatusToDosErrorNoTeb
RtlDllShutdownInProgress
RtlDisownModuleHeapAllocation
NtQueryInformationProcess
Copyright (c) Microsoft Corporation. All rights reserved.
onecore\base\cmd\StartShellExecServiceProvider.h
onecore\base\cmd\maxpathawarestring.cpp
```

xxd

Prints a hexdump of a file

Good to look for recognizable hex patterns or perform advanced hex manipulation



Usage:

```
xxd <FILE>
```

For hex editing: Ghex, or Hexedit

grep

Search for text matches within a file or recursively in files!

```
grep "text you want to find" <FILE>
grep -R "text you want to find" <DIRECTORY>
```

Combine with other utilities!

```
cat <FILE> | grep "text"
strings <FILE> | grep "text"
```

```
cbcicada@DESKTOP-5T74EEM:/mnt/c/Users/CBCicada/Downloads$ strings Goose.dmp | grep uiuctf
uiuctf{W1LD_G0OS3_CH4S3_8234819284901_UNGUESSABLE_1337}
uiuctf{W1LD_G0OS3_CH4S3_8234819284901_UNGUESSABLE_1337}
uiuctf{W1LD_G0OS3_CH4S3_8234819284901_UNGUESSABLE_1337}
uiuctf{NO_THIS_FLAG_AIN'T_IT321}YOU\?_+GITGOOSED-~i>-u
```

Steganography

It sucks

Steganography

- Hide data in other data

- Inherently guessy during CTFs
  - Try lots of ideas
  - Waste lots of time
  - Use statistical approaches if applicable

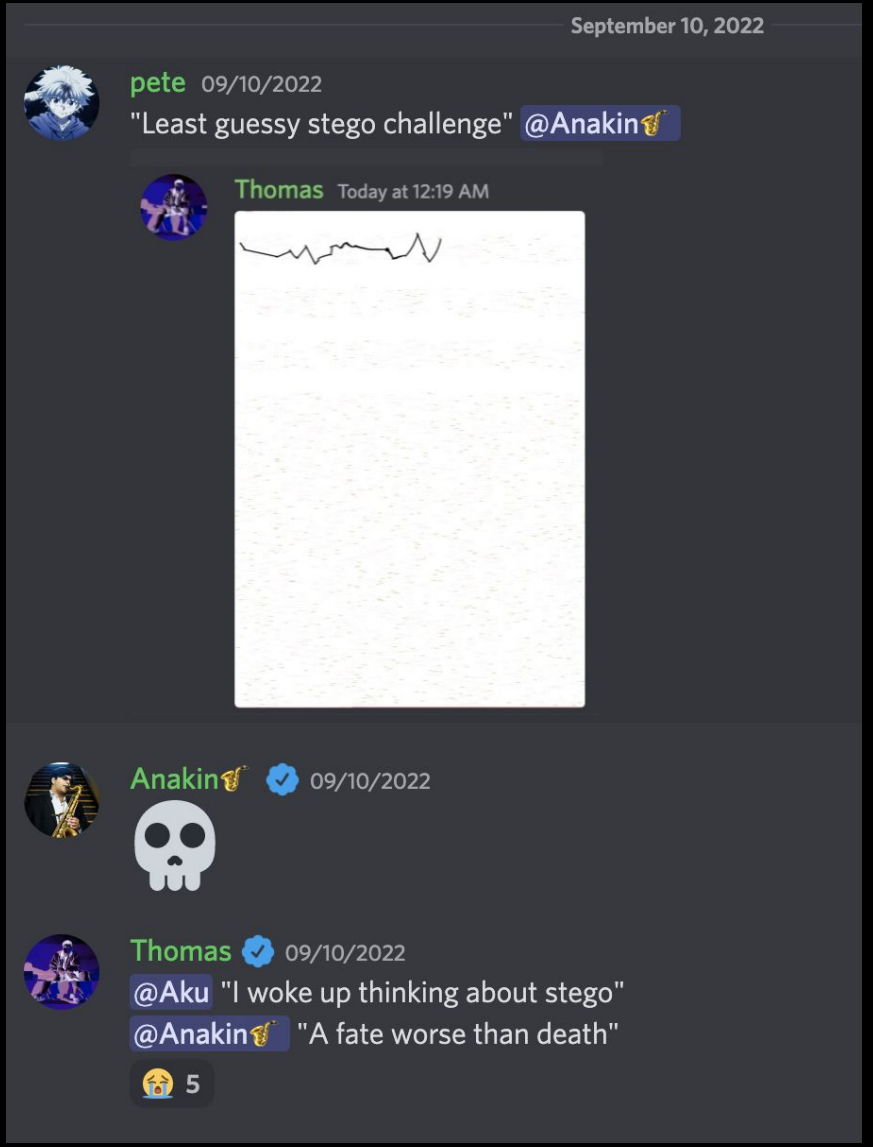
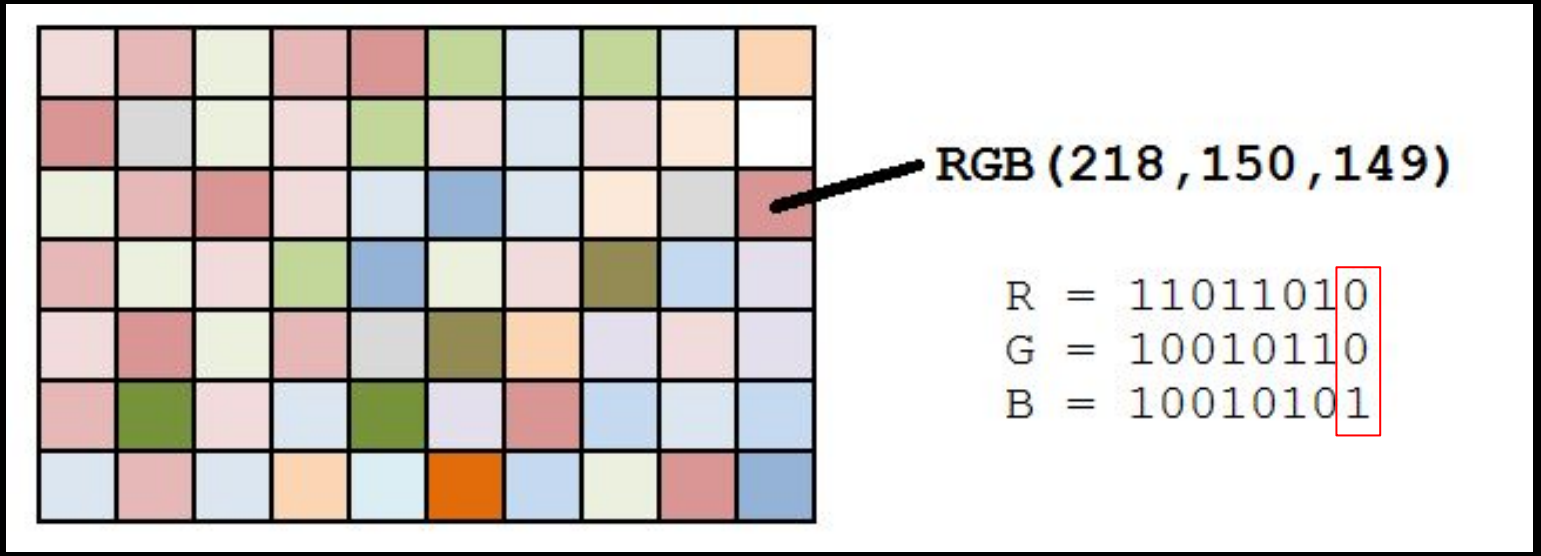


Image Stego - LSB

- LSB (least significant bit) encoding
- Not really useful in the real world, but CTFs love it
- Take the least significant bit (last bit) of each color byte and concatenate all of them to form a message
- Image is mostly visibly unchanged

Image Stego - LSB



Message = (R & 1) || (G & 1) || (B & 1) = 001...

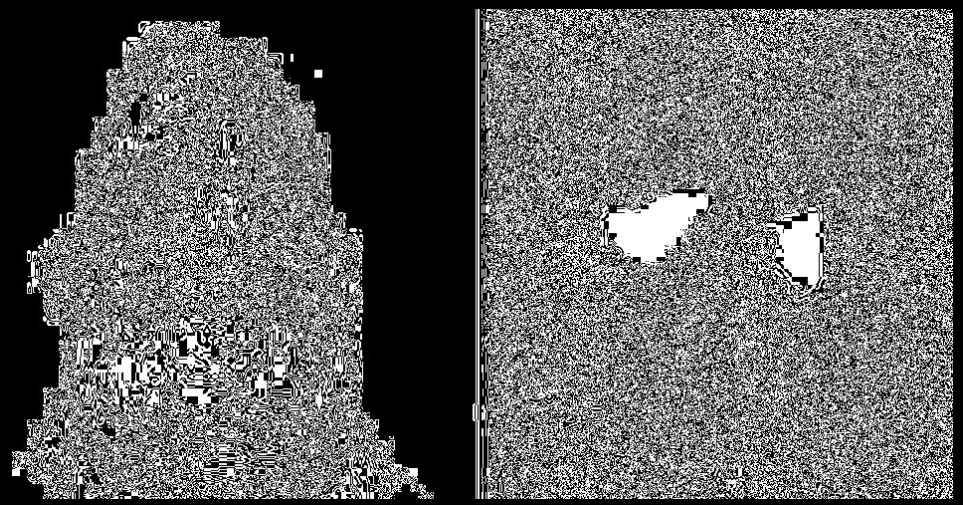
https://towardsdatascience.com/steganography-hiding-an-image-inside-another-77ca66b2acb1

Can you tell the difference?

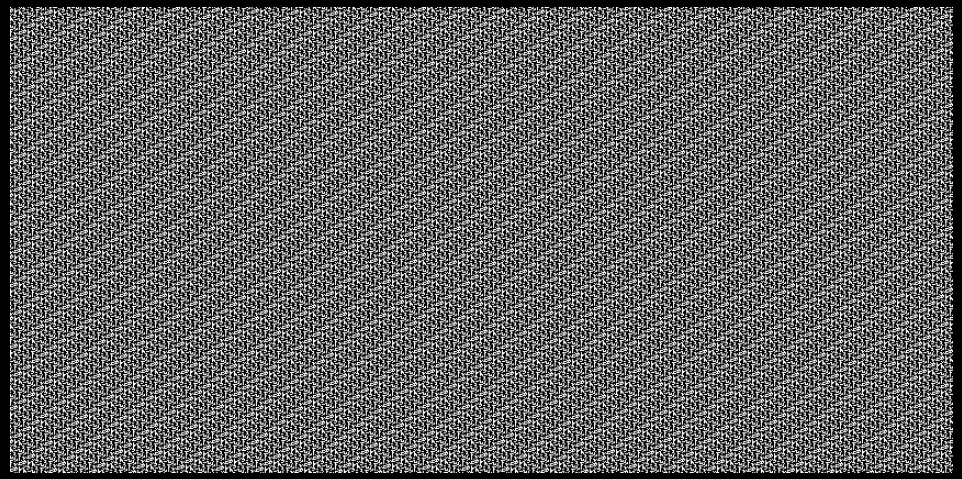
Original

LSB Encoded

Can you tell the difference?



Original (Red, Bit 0)



LSB Encoded (Red, Bit 0)

Stegoanalysis tool: https://stegonline.georgeom.net/image

Image Stego - Malicious Purpose

- Imagine in a red team scenario, I already have control over a machine, but I need to download and execute a binary file.
- The network may be heavily monitored with Deep Packet Inspection, and the file may be inspected by Antivirus before executed
- Plain shellcode will get detected immediately
- Instead, downloading an image that has the secret binary encoded with LSB is way less suspicious

- POC

Other Stego

- Audio stego
  - 90% of the time it is a spectrogram
  - The other 10% is either
    - SSTV
    - Some frequency modulation
    - Some other guessy home-brewed bullshit

  - Most audio stego can be solved with tools
  - You can kidnap an ECE major to solve these stegos

Starting Point

- What kind of file is it?
  - Use a command like `file` or `binwalk` to identify what a file is using magic bytes
  - If it's an image/video/audio, it likely contains hidden information in the form of steganography
  - If it's a document, there might be hidden information in the file format (did you know that a .docx file is just a ZIP archive?)
  - Also if it’s doc, docm, it might contain macros
  - If it's plain text ASCII, what does it contain? Are there any patterns?
  - If it's a binary file, what readable strings are there? Are there patterns in the hexadecimal representation?
- What metadata does it have?
  - Images: location, camera model, encoding
  - Documents: username of creator, directory where saved
  - Everything else

OS Forensics

Log analysis, filesystem/disk

Logs

- Logs provide valuable information about what happened on the system
- You can construct a list of events that occur to determine entry point of an attack and what an attacker did

Linux:

- /var/log/http/access.log
- /var/log/syslog
- other various service logs
- ~/.bash_history or ~/.viminfo if they're really dumb

Windows:

- Event Viewer (Including PS script block logging)
- PSReadline Module
  - (Get-PSReadlineOption).HistorySavePath

See Windows Forensics and Linux Forensics slides from Purple Team for more

Starting Point

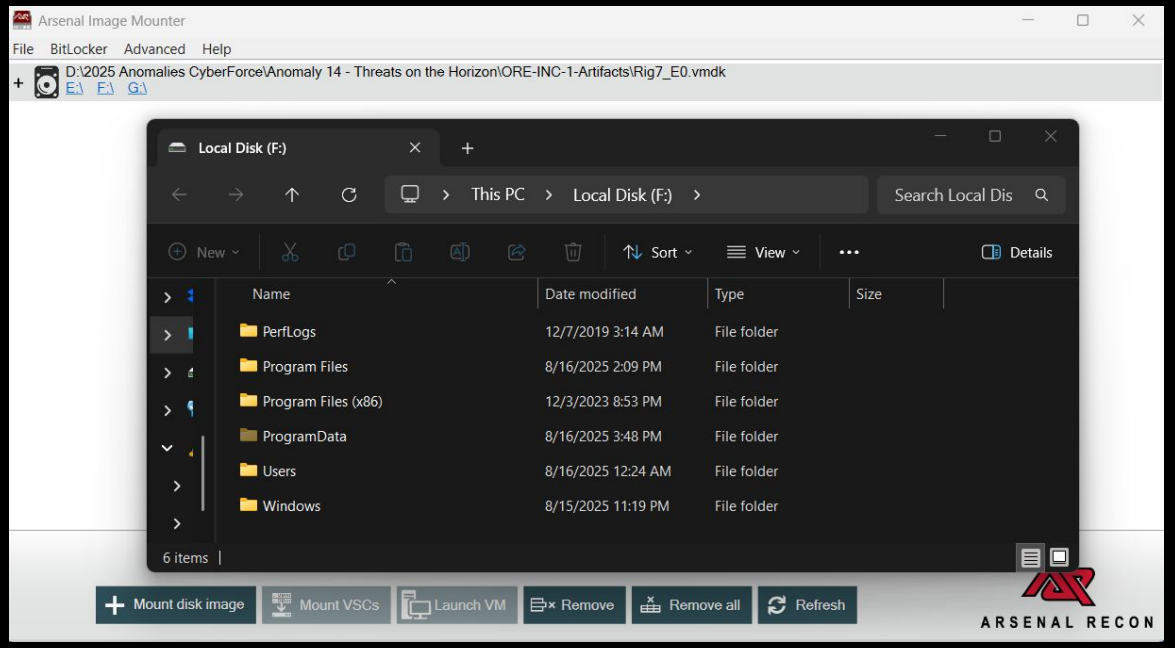
What we are given:

- An archive/zip of the full/partial filesystem
  - We can inspect logs, user files, command history, etc.
- A disk image file (.dd, .iso, .vhd, .vmdk)
  - Follows a filesystem format (e.g. NTFS, ext4)
  - Deleted files can be present and ignored by the filesystem format
  - We can mount it to a live system to examine the contents
- Live access to a system
  - Extremely difficult to perform forensics on a live system while maintaining integrity of evidence since it is volatile
  - Obtain a disk image after attempting to preserve live system evidence

Mounting File Systems

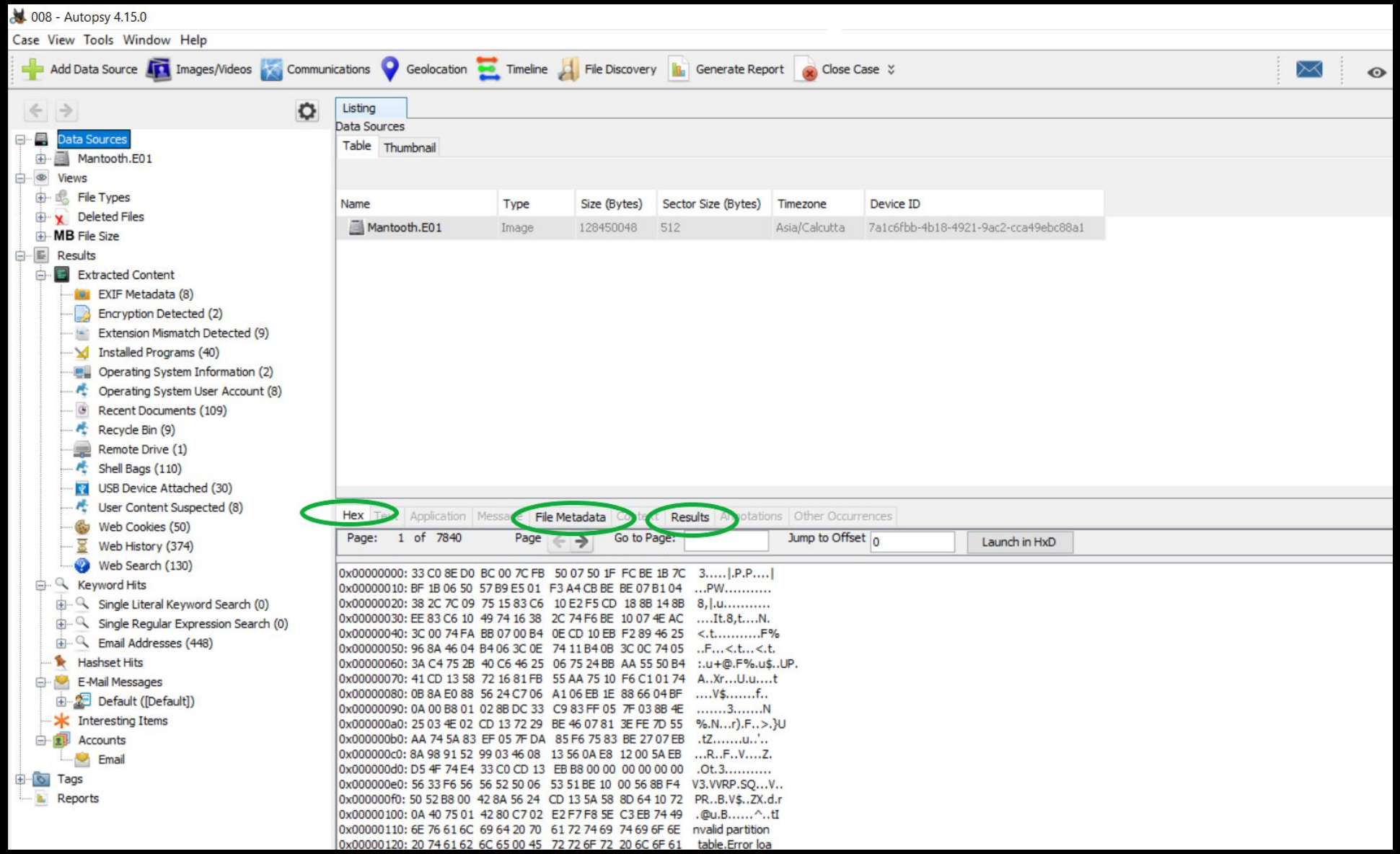
- Mounting
  - Placing the disk image in a usable and accessible state

- Autopsy
- On Windows
  - Arsenal Image Mounter is easy to set up and use (free version too)
- On Linux
  - Use a loop device and mount to mount disk images



Autopsy

- A GUI-based digital forensics platform built on top of The Sleuth Kit (TSK)
- Free, open-source, and runs on Windows, Linux, and macOS
- Key Capabilities
  - Disk & File System Analysis (NTFS, FAT, EXT, APFS, etc.)
  - Timeline Generation of system and user activity
  - Keyword Search and file content analysis
  - Registry, Browser, and Email Artifact Extraction
  - Hash-Based Filtering using NSRL or custom hashsets
  - Image Support: RAW, E01, VHD/VMDK, AFF, and more



Credit: [Geeks for Geeks]

Memory Forensics

Memory Forensics

Instead of being given a traditional filesystem, you are given the contents of memory or RAM in the form of a dump file

- Dump files are usually created when a program crashes or your OS crashes for debugging purposes (or by acquisition tools)
- They can contain sensitive information that was located in memory at the time of crashing or acquisition
  - Passwords in your password manager
  - Clipboard content at time of crash
  - Network connections (possibly to C2 servers)
  - $MFT
  - Loaded DLLs
  - lsass.exe

Volatility

- Analyze memory dump files
- There are two versions of Volatility:
  - Volatility 2.6
    - Is older, but works great when it works
    - Is limited to older OS versions (e.g. doesn't have latest Windows 10 symbols)
    - Intended to be used standalone
  - Volatility 3
    - Complete framework rewrite, under development, and missing some features
    - Is way faster and has better OS version profiling and support
    - Intended to be used as a library with standalone support
    - Annoyingly enough, also has versions such as 2.4.1

Setting up Volatility

```
git clone https://github.com/volatilityfoundation/volatility3.git
cd volatility3
pip install -r requirements.txt
python3 vol.py -f <FILE> <PLUGIN_NAME> (<PLUGIN_OPTION>)
```

```
$ python3 vol.py -f MEMORY.DMP windows.pslist
Volatility 3 Framework 2.4.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime        File output

4       0       System  0xb50a3627b040  137     -       N/A     False   2022-11-17 23:06:49.000000      N/A     Disabled88      4
   Disabled
260     4       smss.exe        0xb50a39093040  2       -       N/A     False   2022-11-17 23:06:49.000000      N/A     Disabled
388     376     csrss.exe       0xb50a36fb8140  12      -       0       False   2022-11-17 23:06:54.000000      N/A     Disabled
460     376     wininit.exe     0xb50a39790080  1       -       0       False   2022-11-17 23:06:54.000000      N/A     Disabled
468     452     csrss.exe       0xb50a397f7140  14      -       1       False   2022-11-17 23:06:54.000000      N/A     Disabled
556     452     winlogon.exe    0xb50a39e4b080  6       -       1       False   2022-11-17 23:06:55.000000      N/A     Disabled
596     460     services.exe    0xb50a39e4a140  8       -       0       False   2022-11-17 23:06:55.000000      N/A     Disabled
612     460     lsass.exe       0xb50a39e5d0c0  11      -       0       False   2022-11-17 23:06:55.000000      N/A     Disabled
708     596     svchost.exe     0xb50a39ecb280  1       -       0       False   2022-11-17 23:06:56.000000      N/A     Disabled
732     556     fontdrvhost.ex  0xb50a39f07180  5       -       1       False   2022-11-17 23:06:56.000000      N/A     Disabled
740     460     fontdrvhost.ex  0xb50a39f09180  5       -       0       False   2022-11-17 23:06:56.000000      N/A     Disabled
748     596     svchost.exe     0xb50a39f0b280  24      -       0       False   2022-11-17 23:06:56.000000      N/A     Disabled
852     596     svchost.exe     0xb50a39fc4300  15      -       0       False   2022-11-17 23:06:56.000000      N/A     Disabled
```

Symbol Tables

- Symbols
  - Volatility must reconstruct kernel data structures from a raw memory image
  - To do this, it needs two critical pieces of information for each kernel object:
    - Offset: Where the structure is located in memory
    - Template / Layout: The structure definition (fields, sizes, types)
- Symbol Table
  - A collection of named kernel objects, each defined by:
    - Symbol name → offset + structure layout
  - In Volatility 3, these come in ISF (Intermediate Symbol File) format (JSON).

Symbol Tables Cont.

- How Volatility Uses Symbols
  - Map virtual → physical memory using Volatility's layered model
  - Instantiate kernel objects (tasks, vm areas, lists, pointers, etc.)
  - Plugins can now:
    - Walk kernel structures
    - Follow pointers
    - Read and cast arbitrary memory
    - Enumerate processes, sockets, modules, etc.

Windows vs. Linux Symbols

- Windows
  - Easy: Microsoft publishes Program Database files (PDBs) with full kernel data.
  - Volatility automatically downloads and parses them.
- Linux
  - Hard: Too many distros, versions, architectures, and configs.
  - Kernel images are stripped by default → no DWARF data.
  - Can use remotely hosted symbols tables as to not download/create a profile for each new kernel/distribution you want to analyze
    - If it is a more niche kernel, you will need to create a new symbol table

```
python3 vol.py --remote-isf-url
'https://github.com/Abyss-W4tcher/volatility3-symbols/raw/master/banners/banners
.json' -f <memory_dump> <plugin>
```

Common Volatility Plugins

- `windows.pstree:` Show parent/child
- `relationshipswindows.psxview`: Cross-view process detection (rootkits)
- `windows.netscan`: Scan for sockets, TCP/UDP connections
- `windows.registry.printkey`: View registry keys/values

- Good Windows Starter (Docs)
- Linux Documentation

Or... just use basic commands

A lot of memory is simply stored as strings!

```
strings <FILE> | grep "text"
```

```
strings --encoding=l <FILE> | grep "text"
```

If you are looking for a specific file format, you can also try grep the file header / signature.

Resources

Windows filesystem artifacts:
https://www.sans.org/posters/windows-forensic-analysis/

(Mirror) https://tinyurl.com/sanswindowsposter

Next Meetings

`2025-12-7` **- Minecraft Social**

- Woo hoo!

`2025-12-19` **- Winter Break**

- Enjoy your break!

ctf.sigpwny.com

`sigpwny{on_the_loose}`

Meeting content can be found at sigpwny.com/meetings.

SIGPwny