Metadata Scanner User Manual

Course:

Cyber Security, May, 2020

Project Name:

#2 - Metadata.

Objective:

Create a code that scans pictures from URLs of a database and finds if there is hidden data and GPS coordinates.

Student Name:

Robert Jonny Tiger.

Metadata Scanner User Manual

Table of Contents

Metadata Scanner User Manual

Metadata Scanner User Manual

Introduction

About Me

My name is Robert Jonny Tiger, I'm a student at John-Bryce in Cyber Security Intelligence course. I'm **NOT** an expert in cyber-security nor presume to be one. I decided to write this manual in a way that even a total beginner can understand how to use the script and work with it.

In part of the course, we are being asked to do various projects and assignments – This is one of them.

The Objective

Create a code that scans pictures from URLs of a database and finds if there is hidden data and GPS coordinates.

This document intends to help non-experienced individuals in understanding how to use the Metadata_Scanner script

Script Workflow

1. Installs Nipe and activates it.
2. Installs ExifTool if you don't have it.
3. Making sure that Nipe is active before starting the script.
4. Creating necessary directories and sub-directories in the parent directory (Project_Metadata)
5. Downloads images from websites of your choice. (You can add URLs as you see fit without stopping the script - See 'URLs.db' file for more info)
6. Scans all images with ExifTool for GPS metadata. If any is found - copies the images to another directory for further analysis.
7. Scans all images with Binwalk for hidden data. If any is found - copies the images to another directory for further analysis.
8. Skips images that he had scanned before, saving time and power.
9. Removes images that it had scanned before.
10. The script creates another script (Continue_After_Reboot.sh) that starts the Metadata_Scanner from the last point it was terminated and in background.

What is TOR?

"Tor is free and open-source software for enabling anonymous communication. The name derived from the acronym for the original software project name "**The Onion Router**"."

Source: Tor (anonymity network) - https://en.wikipedia.org

What is Nipe?

Nipe is a script to make Tor Network your default gateway. It is essential for when you run the script to hide your identity while scanning various websites. The script installs and activates Nipe before making any scan and\or download actions.

What is ExifTool?

"**ExifTool** is a free and open-source software program for reading, writing, and manipulating **image**, audio, video, and PDF metadata."

Source: ExifTool - https://en.wikipedia.org

Metadata Scanner User Manual

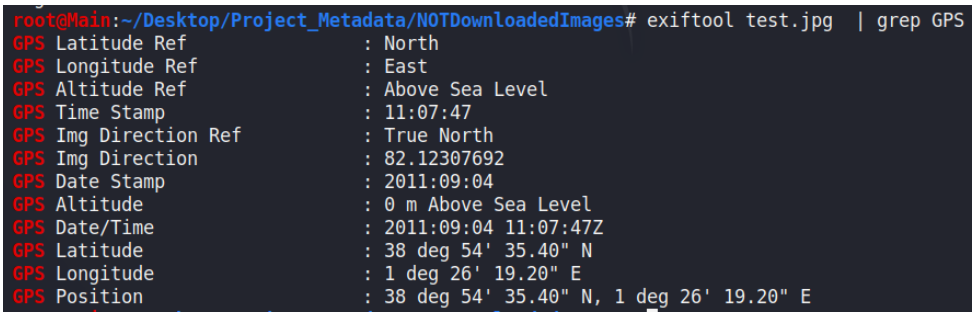


Figure I: ExifTool Scan Example.

The Script will automatically install **ExifTool** if you don't have it.

Using the syntax `exiftool <Image Name> | grep GPS`, ExifTool will parse the image and show us GPS metadata if exists. That's what the script is searching for automatically.

What is Binwalk?

Binwalk is a tool for searching a given binary image for embedded files and executable code. Specifically, it is designed for identifying files and code embedded inside of firmware images.

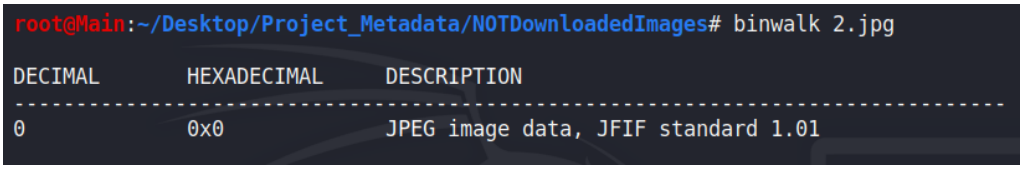


Figure II: Binwalk scan Example.

Here, using the syntax `Binwalk <Image Name>`, Binwalk will parse the image, search for known file headers and output the results to the screen. In this example Binwalk hasn't found any because there are none. It only found a single image – which is the case.

Let's give Binwalk an image that is hidden "behind" another image.

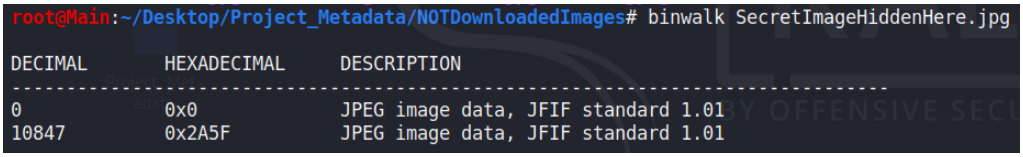


Figure III: Binwalk scan with hidden data example.

In this example Binwalk found 2 images in the binary data of the "SecretImageHiddenHere.jpg" image file. That's what the script is searching for automatically.

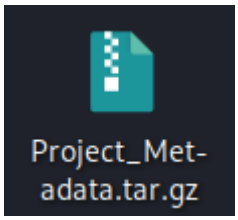
What You Will Need in Order for the Script to Run

- Server running Kali Linux (or other pen testing distribution)
- User with root privileges. (Highest user in the hierarchy)
- 24/7 Internet connection.
- Binwalk.
  Assuming you are working with root privileges, here is a quick installation guide if you don't have it yet:
  1. `apt-get update -y`
  2. `apt-get install -y Binwalk`

- ExifTool.
  Assuming you are working with root privileges, here is a quick installation guide if you don't have it yet (Also, The Script will automatically install **ExifTool** if you don't have it):
  1. **Download** the **Image-ExifTool distribution** from the ExifTool home page
     (The file you download should be named "`Image-ExifTool-12.01.tar.gz`".)
  2. **Unpack the distribution** and **make it your current directory** by typing (each line at a time):
     ```
     cd <your download directory>
     gzip -dc Image-ExifTool-12.01.tar.gz | tar -xf -
     cd Image-ExifTool-12.01
     ```
  3. **Test and install ExifTool** by typing:
     ```
     perl Makefile.PL
     make test
     make install
     ```
     You can now run ExifTool by typing "`exiftool`"

Directories & Files

Project_Metadata.tar.gz

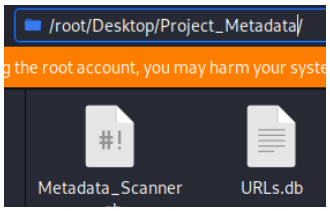


This is the first file you will see. It has the `Metadata_Scanner.sh` and the `URLs.db` file.

- Extract the files to a folder named "Project_Metadata", or simply press Right-Click mouse button → Extract Here.

Figure IV: Project_Metadata Archive.

Project_Metadata



This is the parent \ master folder. All things related to the script and results will be stored here. Nipe will be installed here as well.
Do not change the directory name! The script uses the directory exact name to work properly.

Figure V: Project_Metadata Folder.

Metadata_Scanner.sh

This is the main script. It contains the code to download and scan images recursively from websites based on a URLs database stored in the same directory. **Do not change its name!**

URLs.db

```
# Database of URLs for Metadata_Scanner.sh script.
# ================================================================
# Usage:
# ================================================================
# NOTE: Any line\s with a single hashtag (#) will be ignored by the script.
# 1.  Paste URLs to the top of the file. Starting from line 9. (See 3.)
# 2.  Only full URLs will be scanned (http(or)s://domain.something.something)
# 3.  Enter a new line and paste URLs below:
# ================================================================
# https://thehackernews.com/
https://seekingalpha.com/
```

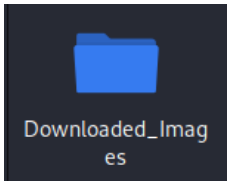
The database in which website's URLs are stored for the script to download. The URLs are stored each URL in a single line with nothing else. Any line with '#' will be ignored by the script.

Figure VI: URLs.db list.

Metadata Scanner User Manual

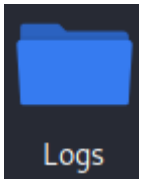
In the example above, thehackernews.com will be ignored by the script. Seekingalpha.com will be scanned. **Do not change its name!**

Downloaded_Images



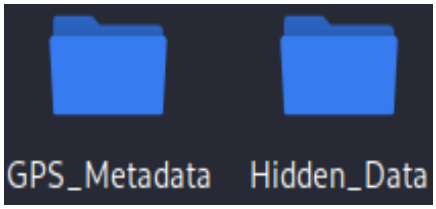
A directory where all images are downloaded to, regardless of which website they got downloaded of. The script removes images that he had scanned before and are no longer necessary from this directory.

Logs



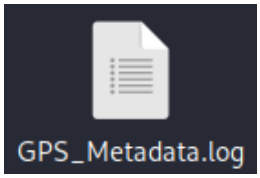
This directory has 2 sub-directories for each of the scans happening in the script:

GPS_Metadata directory will contain images that have GPS metadata stored in them + log file with the image name and the metadata that was found. **All ExifTool related results are saved in here**.



Hidden_Data directory will contain images that are suspicious in having data hidden behind them + log file with the image name and the Binwalk scan result for further analysis. **All Binwalk related results are saved in here**.

GPS_Metadata.log

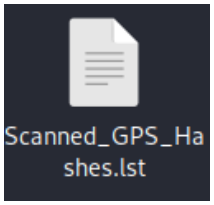


This log file stores **all GPS Metadata** found from images as well as their names and what website they were downloaded of. Example:

```
[!] [FOUND]: GPS_DATA.jpg from https://hamas.ps/ has the following GPS Metadata:
File Name                       : GPS_DATA.jpg
GPS Latitude Ref                : North
GPS Longitude Ref               : East
GPS Altitude Ref                : Above Sea Level
GPS Time Stamp                  : 11:07:47
GPS Img Direction Ref           : True North
GPS Img Direction               : 82.12307692
GPS Date Stamp                  : 2011:09:04
GPS Altitude                    : 0 m Above Sea Level
GPS Date/Time                   : 2011:09:04 11:07:47Z
GPS Latitude                    : 38 deg 54' 35.40" N
GPS Longitude                   : 1 deg 26' 19.20" E
GPS Position                    : 38 deg 54' 35.40" N, 1 deg 26' 19.20" E
[>] [COPIED]: GPS_DATA.jpg to Logs/GPS_Metadata for further analysis.
```

Figure VII: GPS_Metadata.log example.

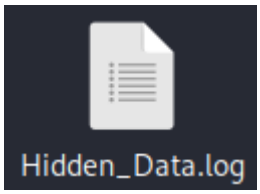
Scanned_GPS_Hashes.lst



This list contains sha256 hashes of images that had been scanned before and\or images that don't have any relevant metadata within them. This list is parsed throughout the script to check whether an image has been scanned before or not. If the image's hash is in the list, the script will remove the image and move on to the next one – saving time and energy.

Metadata Scanner User Manual

Hidden_Data.log



This log file stores possible Hidden Data found from images as well as their names and what website they were downloaded of. Example:

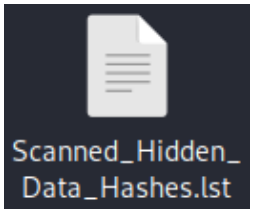
```
[!] [FOUND]: suspicious file: GPS_DATA.jpg from https://hamas.ps/, you might want to inspect it further:

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
30            0x1E            TIFF image data, big-endian, offset of first image directory: 8
1160          0x488           Copyright string: "Copyright (c) 1998 Hewlett-Packard Company"

[>] [COPIED]: GPS_DATA.jpg to Logs/Hidden_Data for further analysis
```

Figure VIII: Hidden_Data.log example.

Scanned_Hidden_Data_Hashes.lst

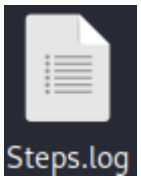


This list contains sha256 hashes of images that had been scanned before and\or images that don't have any relevant metadata within them. This list is parsed throughout the script to check whether an image has been scanned before or not. If the image's hash is in the list, the script will remove the image and move on to the next one – saving time and energy

Nipe

After running the script for the first time, it will install Nipe automatically to the Project's directory. If you don't know what you are doing you should not temper with this directory. Let it stay there.

Steps.log



This file is generated by the script to log at what stage the script stopped for any reason. The script will check with this file to understand where to begin the next time the script runs not for the first time.

Example: (Script completed **BEGINNING** and **ROOT** stages)

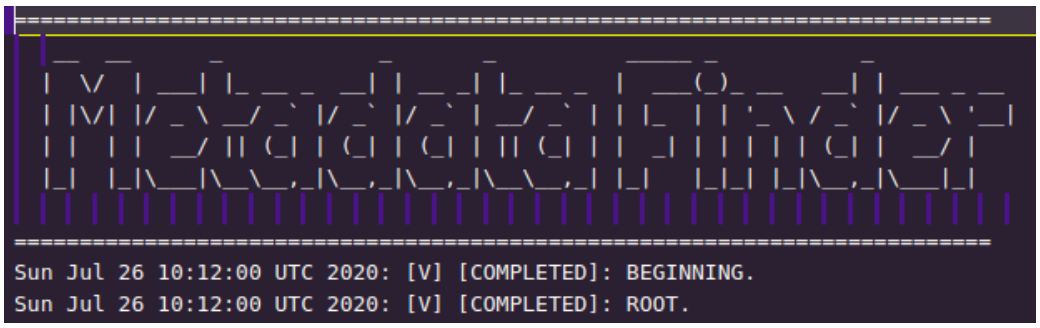
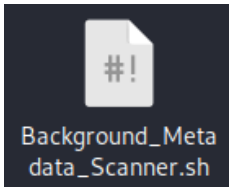


Figure IX: Steps.log example.

Background_Metadata_Scanner.sh



This mini-script will allow you to continue the script after reboot (or any time the script was terminated for some reason) from where it previously stopped, **AND** will start the script in the background. Meaning, you can close the terminal and let the script run. (Using: `Metadata_Scanner.sh &>/dev/null & disown`)

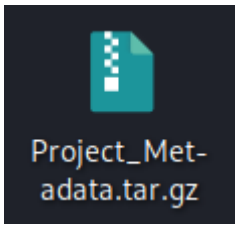
Metadata Scanner User Manual

```
#!/bin/bash
xterm -hold -T Metadata\ Scanner -e echo 'Welcome Back!
The script will start itself and resume the scan in the background as soon as you close the terminal.
Good to Know:
1.  Command: "ps aux | grep Metadata_Scanner" = Check what process #ID the script got in the running processes list.
2.  Command: "kill <process #ID>" = Terminate the process and stop the scan.
3.  View the Logs directory to check progress.
To resume the scan, you may close the terminal now.'
bash /root/Desktop/Project_Metadata/Metadata_Scanner.sh &>/dev/null & disown
```

Figure X: Continue_After_Reboot.sh code.

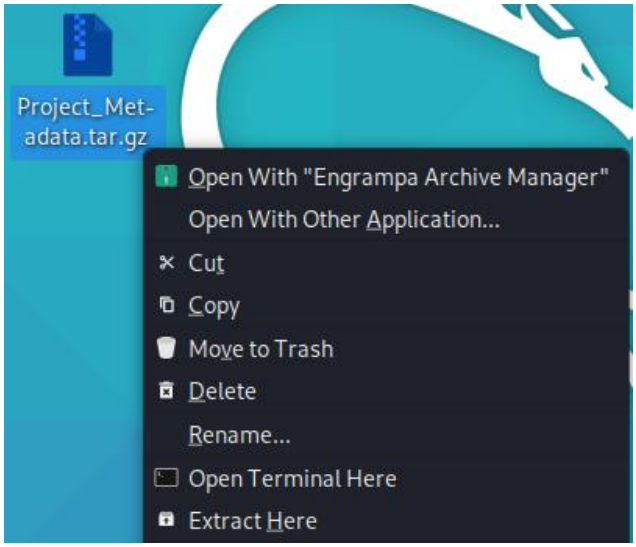
Starting the Script From 0

If you have the tar.gz archive you can start running like so:

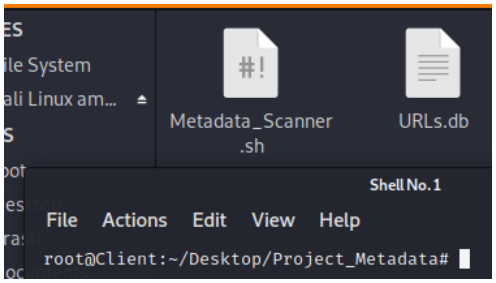


This is the first file you will see. It has the `Metadata_Scanner.sh` and the `URLs.db` file.

1. Extract the files to a folder named “Project_Metadata”, or simply press Right-Click mouse button → Extract Here.



2. Open a terminal in the Project_Metadata directory



3. Run the script as you would with any other script.
`bash <FULL PATH TO SCRIPT>` or `./<SCRIPT NAME>`
Example: `bash /root/Desktop/Project_Metadata/Metadata_Scanner.sh` or `./Metadata_Scanner.sh` (While in Project_Metadata directory)

4. Read the beginning part and then let the script do its magic.

5. If you wish to run the script in the background - start the `Background_Metadata_Scanner.sh` script. This will make the entire script run in the background, 24/7 until the machine shuts-down or the script's process gets terminated for whatever reason.

Metadata Scanner User Manual

Breaking Down the Source Code

Code Intro and Objective Setting

```
1  #!/bin/bash
2  # ================================================================================================================
3  # Course:
4  #   Cyber Security, May, 2020
5  # Project Name:
6  #   #2 - Metadata.
7  # Objective:
8  #   Create a code that scans pictures from URLs of a database and finds if there is hidden data and GPS coordinates.
9  # Student Name:
10 #   Robert Jonny Tiger.
11 # ================================================================================================================
12 # Credits:
13 #   Nipe: GouveaHeitor @ GitHub
14 # ================================================================================================================
```

Figure XI: Lines 1-14: Basic information, The objective of the script & Credits.

Setting Basic Variables

```
16 # Basic variables:
17 echo "Loading... Please hold."
18 DIRPATH=$(find / -type d -name "Project_Metadata" 2>/dev/null) # Sets the location of the script's directory.
19 SCRIPT=Metadata_Scanner.sh # Sets the script's name.
20 DOWNLOADEDIMAGES=$DIRPATH/Downloaded_Images # Sets the Downloaded Images directory.
21 STEPS=$DIRPATH/Steps.log # Sets the completed steps log file.
22 DB=$DIRPATH/URLs.db # Sets the database location.
23 PWD=$(pwd) # Sets the current working directory as variable.
24 USER=$(whoami) # Sets the current user as variable.
```

Figure XII: Lines 16-24: Basic variables in order for the script to run.

Check Where to Start Before Running the Script

```
216 # Checking at what stage the script ended last time and continuing:
217 if [[ -f $STEPS ]]; then # If the Steps.log file exists - do commands:
260 else # If Steps.log not found - do commands: (This is the actual starting point from 0)
261   BEGINNING
262   echo "Cheking if ExifTool is Installed.
263 If not - it will be installed automatically."
264   sleep 3
265   exiftool && echo "[V] ExifTool is Installed. Continuing..." || EXIFTOOL # Run ExifTool. If command is available - move on. If command outputs an error - runs the EXIFTOOL
      function.
266   ROOT
267   REBOOT
268   NIPE
269   ANONYMITY
270   LOGSMAKER
271   SCAN
272 fi
```

Figure XIII: Lines 216-271: Checking where to start the script.

If you run the script for the first time ever, it actually starts in line no. 217. The if statement checks if the `Steps.log` file exists. **If not** – it will execute the functions and commands presented above.

Metadata Scanner User Manual

Functions

BEGINNING

```
93  # Introduction of the script:
94  function BEGINNING {
95    echo "========================================================================" | tee -a $STEPS
96    figlet "Metadata Finder" -c | tee -a $STEPS
97    echo "========================================================================" | tee -a $STEPS
98    echo "$(date -u): [>] [STARTED]: BEGINNING." | tee -a $STEPS
99    tput bold && echo "The following script does the following:" && tput sgr0
100   echo "1.  Installs Nipe (Anonymity) and activates it.
101 2.  Installs ExifTool if you don't have it.
102 3.  Making sure that Nipe is active before starting the script.
103 4.  Creating necessary directories and sub-directories in the parent directory (Project_Metadata)
104 5.  Downloads images from websites of your choice. (You can add URLs as you see fit without stopping the script - See 'URLs.db' file for more info)
105 6.  Scans all images with Exiftool for GPS metadata. If any is found - copies the images to another directory for further analysis.
106 7.  Scans all images with Binwalk for hidden data. If any is found - copies the images to another directory for further analysis.
107 8.  Skips images that it had scanned before, saving time and power.
108 9.  Removes images that it had scanned before.
109 10. The script creates another script (Continue_After_Reboot.sh) that starts the Metadata_Scanner from the last point it was terminated and in background.
110 For more information about how to properly use and work with the script - see 'User Manual.pdf' file."
111   sleep 20
112   tput bold && echo "Would you like to start or exit? (Type 1 or 2)" && tput sgr0
113   select yn in "Start" "Exit"; do
114       case $yn in
115           Start ) break;;
116           Exit ) exit;;
117       esac
118   done
119   echo "$(date -u): [V] [COMPLETED]: BEGINNING." | tee -a $STEPS
120   sleep 4
121 }
```

Figure XIV: Lines 93-121: BEGINNING function.

The **BEGINNING** function is like the homepage of the script. It has the banner and describes what the script does. The user is ought to read the whole page and afterwards it will prompt him the option to start the script or exit.

EXIFTOOL

```
122 # ExifTool installation:
123 function EXIFTOOL {
124   cd $HOME # Installation directory.
125   echo "Downloading and Installing ExifTool..."
126   sleep 3
127   wget https://exiftool.org/Image-ExifTool-12.01.tar.gz
128   gzip -dc Image-ExifTool-12.01.tar.gz | tar -xf -
129   cd Image-ExifTool-12.01
130   perl Makefile.PL
131   make test
132   make install
133   rm -rf Image-ExifTool-12.01* # Removes any installation files leftovers.
134   cd $PWD
135 }
```

Figure XV: Lines 122-135: EXIFTOOL function.

The **EXIFTOOL** function is called if the script finds out that **ExifTool** is **not** installed on the machine. This function will download **ExifTool** installation in the HOME directory and initiate the installation automatically. Afterwards, it will remove the installation files leftovers.

ROOT

```
137 # Run only with root privileges user statement:
138 function ROOT {
139   tput bold && echo "The script runs only with root privileges user." && tput sgr0
140   echo "If you are not using root privileges user the script will exit on it's own."
141   sleep 3
142   if [[ ! $(perl -n -e '@user = split /:/ ; print "@user[0]\n" if @user[2] == "0";' < /etc/passwd) =~ "$USER" ]]; then
      then - do commands.
143     echo "Not running with root privileges. Log in with root privileges user and try again."
144     echo "Exiting..."
145     sleep 3
146     exit
147   fi
148   echo "$(date -u): [V] [COMPLETED]: ROOT." | tee -a $STEPS
149   sleep 3
150 }
```

Figure XVI: Lines 137-150: ROOT function.

The script can only run with **root user privileges**. This function will determine if you can use the script or not. The function gets the list of users that have root privileges and compares it with the current user (`$USER`). If the current user is not in that list, then he doesn't have root privileges, meaning he can't run the script. The script will exit automatically and print "`Not running with root privileges. Log in with root privileges user and try again.`"

Metadata Scanner User Manual

REBOOT

```
152  # Creating a sub-script to make it run in the background:
153  function REBOOT {
154    touch $DIRPATH/Background_Metadata_Scanner.sh # Creats empty file.
155    echo '#!/bin/bash' >> $DIRPATH/Background_Metadata_Scanner.sh
156    echo "xterm -hold -T Metadata\ Scanner -e echo 'Welcome Back!
157  The script will start itself and resume the scan in the background as soon as you close the terminal.
158  Good to Know:
159  1.  Command: \"ps aux | grep Metadata_Scanner\" = Check what process #ID the script got in the running processes list.
160  2.  Command: \"kill <process #ID>\" = Terminate the process and stop the scan.
161  3.  View the Logs directory to check progress.
162  To resume the scan, you may close the terminal now.'" >> $DIRPATH/Background_Metadata_Scanner.sh
163    echo "bash $DIRPATH/$SCRIPT &>/dev/null & disown" >> $DIRPATH/Background_Metadata_Scanner.sh
164    chmod a+rx $DIRPATH/Background_Metadata_Scanner.sh
165    echo "[+] [CREATED]: Background_Metadata_Scanner.sh file. Run it to continue the script after reboot."
166    echo "$(date -u): [V] [COMPLETED]: REBOOT." | tee -a $STEPS
167    sleep 3
168  }
```

Figure XVII: Lines 152-168: REBOOT function.

In order for the script to run in the background regardless of us closing the terminal, this function will create a sub-script that will do the job. The sub-script is called `Background_Metadata_Scanner.sh` and it consists of 2 commands only, "`echo`" and "`bash`" (To run the script) First, it will open an xterm with a message and some good to know information. Then, after closing the xterm the script will run by itself in the background. To check if it's true, run the following command: `ps aux | grep Metadata_Scanner`. This command will show you the processes running on the machine with the name "Metadata_Scanner" in them. You can decide to kill the process with the "`kill`" command followed by the process number.

NIPE

```
170  # Nipe Installation:
171  function NIPE {
172    echo "Installing Nipe (for Anonymity)..."
173    sleep 3
174    # Cloning Nipe to current working directory and cd to nipe:
175    export PERL_MM_USE_DEFAULT=1
176    cd $DIRPATH
177    git clone -q https://github.com/GouveaHeitor/nipe
178    NIPEDIR=$(find $DIRPATH -type d -name nipe 2>/dev/null) # Nipe's Directory.
179    cd $NIPEDIR
180
181    # Installs libs and dependencies:
182    cpan install Try::Tiny Config::Simple JSON
183
184    # Nipe installation:
185    cd $NIPEDIR
186    perl nipe.pl install
187    echo "[+] [INSTALLED]: Nipe."
188    echo "$(date -u): [V] [COMPLETED]: NIPE." | tee -a $STEPS
189    cd $PWD
190    sleep 3
191  }
```

Figure XVIII: Lines 170-191: NIPE function.

When we scan and download data from websites recursively like in this script, we want to **hide our identity** as much as possible. For this, I chose to use **Nipe**. The **NIPE** function will install **Nipe** on your machine.

ANONYMITY

```
193  # Anonymity check before running the script:
194  function ANONYMITY {
195    NIPEDIR=$(find $DIRPATH -type d -name nipe 2>/dev/null) # Nipe's Directory.
196    echo "Checking if Nipe is activated... If not - i will do it for you..."
197    until [[ $(cd $NIPEDIR && perl nipe.pl status | grep Status | awk '{print $3}') == "activated." ]]; do
198      cd $NIPEDIR
199      ./nipe.pl restart
200      cd $PWD
201    done
202    echo "[>] [STARTED]: Nipe."
203    echo "$(date -u): [V] [COMPLETED]: ANONYMITY." | tee -a $STEPS
204    sleep 3
205  }
```

Figure XIX: Lines 193-205: ANONYMITY function.

This function will check whether Nipe is running or not. The script will **not continue** until Nipe is activated. The function sets the Nipe directory first as a variable (`$NIPEDIR`). Then, using the commands `cd $NIPEDIR` and `perl`

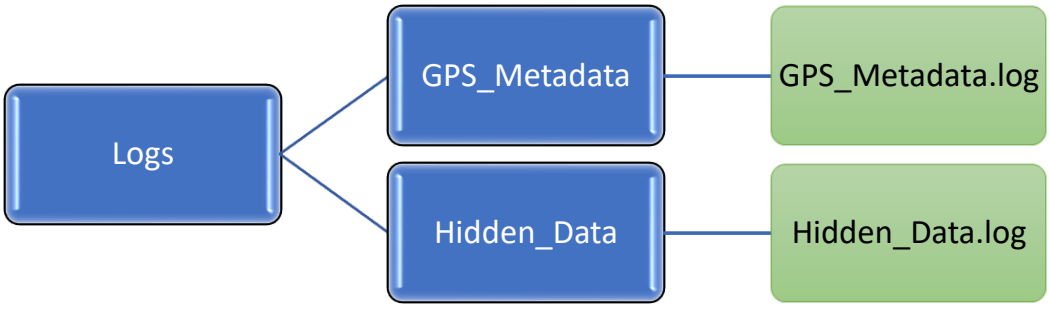
`nipe.pl status` it checks if the status is "activated". If it is, the script moves on, but **until then**, it restarts the Nipe service (Until status = "activated.")

LOGSMAKER

```
207  # Creating logs files and folders:
208  function LOGSMAKER {
209    mkdir $DIRPATH/Logs $DIRPATH/Logs/GPS_Metadata $DIRPATH/Logs/Hidden_Data && touch $DIRPATH/Logs/GPS_Metadata/GPS_Metadata.log $DIRPATH/Logs/Hidden_Data/Hidden_Data.log
210    echo "[+] [CREATED]: 'Logs' directory.
211  [+] [CREATED]: 'GPS_Metadata' and 'Hidden_Data' directories under 'Logs'.
212  [+] [CREATED]: 'GPS_Metadata.log' and 'Hidden_Data.log' log files."
213    echo "$(date -u): [V] [COMPLETED]: LOGSMAKER." | tee -a $STEPS
214    sleep 3
215  }
```

Figure XX: Lines: 207-215: LOGSMAKER function.

This function creates the necessary **Logs** directory, **sub-directories** and **log files** that you will work with. (More info on each one in **Files & Directories** chapter)



Metadata Scanner User Manual

Check From Where to Continue

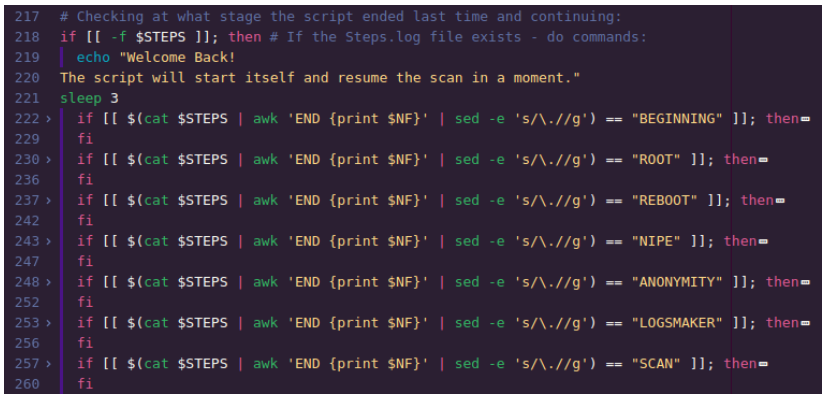


Figure XXI: Lines: 217-260: Check Steps.log.

If you ran the script **at least once** than you will notice a "Steps.log" file was created. This file is in charge of logging where the script ended in the previous time it ran. Upon completing every function, the script injects a single line stating that the stage has been completed with the function's name. When running the script for the second time (or third or fourth and so on..) the script checks with Steps.log where the script ended the last time by grasping the last line and the last word (which is the function's name). For example: If the script reads that the last line and word in Steps.log is "**REBOOT**", it will skip any prior stages to **REBOOT** and continue with **NIPE**, **ANONYMITY**, **LOGSMAKER** and finally **SCAN**.

The Endless Scan

The **SCAN** function is the most **important** part and the **primary objective** of this script. The whole script is built around that function and any other functions and commands are there to assist the **SCAN** function.

Downloading the Images

Figure XXII: Lines 26-35: wget command in while true loop.

The whole download and scan process occurs in a `while true` loop – meaning it's an **infinte** loop, **running 24/7**.

`for FULLURL in $(cat $DB | grep -v '#'); do` Grasping the full URL from the URLs.db file and does the following:

```
wget -e robots=off -nd -r -q --mirror --no-cookies --level=inf --no-check-certificate --no-cache -T 30 --ignore-length -np -P $DOWNLOADEDIMAGES -A
jpeg,jpg,bmp,gif,png,webp,exif,tiff,webp,heif,bat $FULLURL 2>/dev/null # (-e: Execute - no robots file, -nd: No Directories, -r: Recursive, -q: Qui
```

Figure XXIII: Full wget command.

`wget` is the command to download data from websites. Using the **`-A`** flag I determine that only files with image file extensions will be (**A**)ccepted to be downloaded. Other flags used:

- **`-e`**: Execute - no robots file.
- **`-nd`**: No Directories.
- **`-r`**: Recursive (Run through all website's pages recursively)
- **`-q`**: Quiet, no output.
- **`--level`**: Recursive depth `level=infinite`. (By default, recursive depth is set to 5)

- **`--no-check-certificate`**.
- **`--no-cache`**.
- **`-T`**: Timeout <secs>.
- **`--ignore-length`**.
- **`-np`**: No accend to Parent directory.
- **`-P`**: Prefix location.

Checking if an Image Got Scanned Before

```
37      # Checks if images scanned before. If yes - removes them.
38      for IMAGE in $(ls $DOWNLOADEDIMAGES); do
39        HASH1=$(sha256sum $DOWNLOADEDIMAGES/$IMAGE | awk '{print $1}' 2>/dev/null) # Sets sha256 hash variable.
40        HASHINFILE1=$(cat $DIRPATH/Logs/GPS_Metadata/Scanned_GPS_Hashes.lst 2>/dev/null | grep -c $HASH1) # Greps the count of how many times the hash occures in the
          Scanned_GPS_Hashes.log file.
41        if [[ $HASHINFILE1 -eq 1 ]]; then
42          rm $DOWNLOADEDIMAGES/$IMAGE # Removes the image.
43        fi
44      done
```

For every image there is a unique **hash**. When an image gets scanned, its **hash** is stored in a list. In order to not scan the same image twice, the script will parse through the list to check if the image's **hash exists** in that list. If it does the image gets automatically **removed** and not get scanned again. Using **`sha256sum`**.

ExifTool Scan

```
54      # Exiftool scan:
55      for IMAGE in $(ls $DOWNLOADEDIMAGES); do # For every image - do commands:
56        HASH1=$(sha256sum $DOWNLOADEDIMAGES/$IMAGE | awk '{print $1}') # Sets sha256 hash variable.
57        HASHINFILE1=$(cat $DIRPATH/Logs/GPS_Metadata/Scanned_GPS_Hashes.lst 2>/dev/null | grep -c $HASH1) # Greps the count of how many times the hash occures in the
          Scanned_GPS_Hashes.log file.
58        if [[ $HASHINFILE1 -eq 0 ]]; then # If the hash never occured in the file - do commnads.
59          if [[ $(exiftool $DOWNLOADEDIMAGES/$IMAGE | grep 'GPS\|File Name' | wc -l) -gt "1" ]]; then # If line count is > 1 - do commands.
60            echo "[!] [FOUND]: $IMAGE from $FULLURL has the following GPS Metadata:" >> $DIRPATH/Logs/GPS_Metadata/GPS_Metadata.log # Logs the file name in the log.
61            exiftool $DOWNLOADEDIMAGES/$IMAGE | grep 'GPS\|File Name' >> $DIRPATH/Logs/GPS_Metadata/GPS_Metadata.log # Outputs the exiftool output with just the File Name &
              GPS info to the log.
62            cp $DOWNLOADEDIMAGES/$IMAGE $DIRPATH/Logs/GPS_Metadata # Copies the suspicious image to a folder.
63            echo "[>] [COPIED]: $IMAGE to Logs/GPS_Metadata for further analysis." >> $DIRPATH/Logs/GPS_Metadata/GPS_Metadata.log
64            echo "" >> $DIRPATH/Logs/GPS_Metadata/GPS_Metadata.log # Just to seperate the outputs.
65            sha256sum $DOWNLOADEDIMAGES/$IMAGE | awk '{print $1}' >> $DIRPATH/Logs/GPS_Metadata/Scanned_GPS_Hashes.lst # Logs the sha256 hash to the Scanned_GPS_Hashes.lst
              file.
66          else
67            sha256sum $DOWNLOADEDIMAGES/$IMAGE | awk '{print $1}' >> $DIRPATH/Logs/GPS_Metadata/Scanned_GPS_Hashes.lst # Logs the sha256 hash to the Scanned_GPS_Hashes.lst
              file.
68          fi
69        fi
70      done # End for IMAGE loop. (exiftool)
```

Figure XXIV: Lines 54-70: ExifTool Scan.

Before every image in the Downloaded_Images folder is scanned, the script checks the hash of that image to see if the image got scanned before. If not – the scan will start. If the hash exists – it will skip the image. The scan targets **GPS metadata**. When ExifTool finds **GPS Metadata**, the script will log that information to GPS_Metadata.log with the image name and source website. Then, it will copy the image to another directory called GPS_Metadata for further analysis. If nothing is found – only the hash of the image gets injected in to the hashes list and the image will be removed in the next cycle.

Metadata Scanner User Manual

Binwalk Scan

```
72          # Binwalk scan:
73          for IMAGE in $(ls $DOWNLOADEDIMAGES); do # For every image - do commands:
74            HASH2=$(sha256sum $DOWNLOADEDIMAGES/$IMAGE | awk '{print $1}') # Sets sha256 hash variable.
75            HASHINFILE2=$(cat $DIRPATH/Logs/Hidden_Data/Scanned_Hidden_Data_Hashes.lst 2>/dev/null | grep -c $HASH2) # Greps the count of how many times the hash occures in the
              Scanned_GPS_Hashes.log file.
76            if [[ $HASHINFILE2 -eq 0 ]]; then # If the hash never occured in the file - do commnads.
77              if [[ $(binwalk $DOWNLOADEDIMAGES/$IMAGE | wc -l) -gt "6" ]]; then # If binwalk outputs more than 5 lines than - do commands.
78                echo "[!] [FOUND]: suspicious file: $IMAGE from $FULLURL, you might want to inspect it further:" >> $DIRPATH/Logs/Hidden_Data/Hidden_Data.log # Logs the suspicious
                  file in the log.
79                binwalk $DOWNLOADEDIMAGES/$IMAGE >> $DIRPATH/Logs/Hidden_Data/Hidden_Data.log # Outputs the binwalk output to the log.
80                cp $DOWNLOADEDIMAGES/$IMAGE $DIRPATH/Logs/Hidden_Data # Copies the suspicious image to a folder.
81                echo "[>] [COPIED]: $IMAGE to Logs/Hidden_Data for further analysis" >> $DIRPATH/Logs/Hidden_Data/Hidden_Data.log
82                echo "" >> $DIRPATH/Logs/Hidden_Data/Hidden_Data.log # Just to seperate the outputs.
83                sha256sum $DOWNLOADEDIMAGES/$IMAGE | awk '{print $1}' >> $DIRPATH/Logs/Hidden_Data/Scanned_Hidden_Data_Hashes.lst # Logs the sha256 hash to the
                  Scanned_Hidden_Data_Hashes.lst file.
84              else
85                sha256sum $DOWNLOADEDIMAGES/$IMAGE | awk '{print $1}' >> $DIRPATH/Logs/Hidden_Data/Scanned_Hidden_Data_Hashes.lst # Logs the sha256 hash to the
                  Scanned_Hidden_Data_Hashes.lst file.
86              fi
87            fi
88          done # End for IMAGE in loop. (binwalk)
```

Figure XXV: Lines: 72-88: Binwalk Scan.

Before every image in the Downloaded_Images folder is scanned, the script checks the hash of that image to see if the image got scanned before. If not – the scan will start. If the hash exists – it will skip the image. The scan targets multiple file headers in an image (**Hidden Data**). if Binwalk gets more than 6 lines of output, it will mark the image as **suspicious** for **possibly having hidden data** and will inject the output to Hidden_Data.log with the image name and source website as well as copy the image to Hidden_Data directory for further analysis. If nothing is found – only the hash of the image gets injected to the hashes list and the image will be removed in the next cycle.