Steganography Detection and File Analysis Automation in Python

M. Waqas (24109112)
M. Sohaib (24109111)

Steganography Detection and File Analysis Automation in Python

1. Objective

This project automates the process of detecting and analyzing hidden content in files (typically images) using steganographic and forensic techniques. The Python script utilizes command-line tools to extract metadata, hidden files, and even brute-force steganographically hidden content.

2. Tools & Technologies Used

Command-line tools integrated with the script:

- `exiftool`: Reads file metadata.
- `strings`: Extracts readable strings from binary files.
- `binwalk`: Analyzes binary files for embedded data.
- `foremost`: Recovers deleted/hidden files from binary files.
- `steghide`: Extracts data hidden with steganography.
- `rockyou.txt`: A common wordlist used for password brute-forcing.

3. Code Workflow Explanation

Your Python script consists of several functional blocks:

1. **Metadata Extraction (`exiftool`)**
   - Command: `exiftool <file>`
   - Extracts author, creation date, software used, etc.
2. **String Extraction (`strings`)**
   - Command: `strings <file>`
   - Reveals embedded readable text or signatures.
3. **Embedded File Detection (`binwalk`)**
   - Command: `binwalk -e <file>`
   - Searches for and extracts embedded files or compressed data.
4. **Hidden File Recovery (`foremost`)**
   - Command: `foremost -i <file> -T -o output`
   - Scans for file headers/footers and recovers hidden content.
5. **Brute-Force Steganographic Content (`steghide`)**
   - Iterates over a wordlist to find the password used to hide the content.

4. How to Run the Script on Kali Linux

A. Install Required Dependencies

```
sudo apt update
sudo apt install exiftool binwalk foremost steghide stegosuite
sudo apt install binutils  # for 'strings' command
```

B. Prepare Script and File

- Place your Python script (e.g., `steg_analysis.py`) and test image (e.g., `secret.jpg`) in the same folder like `/home/user/`.
- Save your script:
- `nano steg_analysis.py`

  Paste the code and press `Ctrl+X`, then `Y`, then `Enter`.

C. Make It Executable

```
chmod +x steg_analysis.py
```

D. Run the Script

```
python3 steg_analysis.py
```

E. Provide Inputs When Prompted

Example inputs during execution:

- Enter path: `/home/user/secret.jpg`
- Want to brute-force password? `y`
- Enter wordlist: `/usr/share/wordlists/rockyou.txt`

5. Run Tools Manually from Shell (Alternate Option)

If you want to test commands individually:

```
cd /home/user

exiftool secret.jpg
strings secret.jpg | less
binwalk secret.jpg
binwalk --extract secret.jpg
foremost -i secret.jpg -T -o output
steghide extract -sf secret.jpg -p password123
```

6. Output & Interpretation

- **ExifTool Output:** Shows file metadata.
- **Strings Output:** Can contain hidden messages or clues.
- **Binwalk Output:** Lists embedded files and where they're located.
- **Foremost Output:** Recovers files into `/output` folder.
- **Steghide Output:** If password is correct, reveals hidden file.

7. Brute-force Steghide Results

- If password is found using `rockyou.txt`, a file named `extracted_secret.txt` will appear in your folder.

8. Conclusion

This assignment demonstrates how a Python script can integrate multiple forensic tools for automated steganography detection and analysis. It reduces manual errors, speeds up investigation, and helps beginners understand how each tool works in practice.

Annexure-I

Python Code:

```
import os
import subprocess
def run_exiftool(file_path): print("\n[+] Running ExifTool:") try: result =
subprocess.run(["exiftool", file_path], capture_output=True, text=True)
print(result.stdout) except Exception as e:
print(f"[-] Error running exiftool: {e}") def run_strings(file_path):
print("\n[+] Extracting strings:") try: result = subprocess.run(["strings",
file_path], capture_output=True, text=True)
print(result.stdout) except Exception as e:
print(f"[-] Error running strings: {e}") def run_binwalk(file_path):
print("\n[+] Running binwalk -e to extract embedded files:") try:
subprocess.run(["binwalk", "-e", file_path]) except Exception as e:
print(f"[-] Error running binwalk: {e}") def run_foremost(file_path):
print("\n[+] Running foremost to recover hidden files:") try: output_dir =
"output" subprocess.run(["foremost", "-i", file_path, "-T", "-o",
output_dir])
print(f"[+] Foremost results saved to ./{output_dir}/") except Exception as
e:
print(f"[-] Error running foremost: {e}") def
brute_force_steghide(file_path, wordlist_path): print("\n[+] Brute-forcing
steghide password (this may take time)...") try: with open(wordlist_path,
"r", encoding="latin-1") as f: passwords = f.read().splitlines()
for pw in passwords:
result = subprocess.run( ["steghide", "extract", "-sf", file_path, "-p",
pw, "-xf", "extracted_secret.txt"], capture_output=True, text=True )
if "wrote extracted data to" in result.stdout.lower(): print(f"[+] Password
found: {pw}")
print("[+] Extracted file: extracted_secret.txt") return
print("[-] Password not found in wordlist.") except Exception as e:
print(f"[-] Error in brute-force: {e}") def main(): file_path =
input("Enter path to stego file (e.g., /home/user/stego.jpg): ").strip() if
not os.path.isfile(file_path): print("[-] File not found.") return
run_exiftool(file_path) run_strings(file_path) run_binwalk(file_path)
run_foremost(file_path) choice = input("\nDo you want to brute-force
steghide password? (y/n): ").strip().lower() if choice == 'y':
wordlist_path = input("Enter path to wordlist (e.g.,
/usr/share/wordlists/rockyou.txt): ").strip() if
os.path.isfile(wordlist_path): brute_force_steghide(file_path,
wordlist_path) else: print("[-] Wordlist file not found.") if __name__ ==
"__main__": main()
```

__ python

```
import os
import subprocess
```

Explanation: These are standard Python modules:

• `os`: Provides functions to interact with the operating system (e.g., checking if a file exists).

• `subprocess`: Allows execution of shell commands from within Python.

python

```
def run_exiftool(file_path): print("\n[+] Running ExifTool:")
```

Explanation: Defines the function `run_exiftool()` to extract metadata from a file using `ExifTool`. It takes `file_path` as an argument and prints a message indicating that the tool is being executed.

python

```
try: result = subprocess.run(["exiftool", file_path], capture_output=True,
text=True) print(result.stdout)
```

Explanation: Uses `subprocess.run()` to execute `exiftool` with the provided file path. The `capture_output=True` ensures output is captured, and `text=True` converts it into a readable string. It then prints the metadata extracted from the file.

python
```
except Exception as e:
print(f"[-] Error running exiftool: {e}")
```
Explanation: Catches any errors that occur while running `exiftool` and prints an error message.
python
```
def run_strings(file_path): print("\n[+] Extracting strings:")
```
Explanation: Defines the `run_strings()` function, which extracts readable strings from the given file.
python
```
try: result = subprocess.run(["strings", file_path], capture_output=True,
text=True) print(result.stdout)
```
Explanation: Runs the `strings` command on the file, capturing readable text that may indicate hidden data or embedded messages.
python
```
except Exception as e:
print(f"[-] Error running strings: {e}")
```
Explanation: Handles errors that may occur while executing the `strings` command. python
```
def run_binwalk(file_path): print("\n[+] Running binwalk -e to extract
embedded files:")
```
Explanation: Defines `run_binwalk()` to search for embedded files within the provided file using `binwalk`.
python
```
try: subprocess.run(["binwalk", "-e", file_path])
```
Explanation: Executes `binwalk` with the `-e` flag, which attempts to extract hidden embedded files.
python
```
except Exception as e:
print(f"[-] Error running binwalk: {e}")
```
Explanation: Handles errors that may occur when running `binwalk`. python
```
def run_foremost(file_path): print("\n[+] Running foremost to recover
hidden files:")
```
Explanation: Defines `run_foremost()`, which recovers hidden or deleted files using the `foremost` tool.
python
```
try: output_dir = "output" subprocess.run(["foremost", "-i", file_path, "-
T", "-o", output_dir]) print(f"[+] Foremost results saved to
./{output_dir}/")
```
Explanation: Runs `foremost` on the file, saving recovered files to the "output" directory.
python
```
except Exception as e:
print(f"[-] Error running foremost: {e}")
```
Explanation: Handles errors that occur when executing `foremost`. python
```
def brute_force_steghide(file_path, wordlist_path): print("\n[+] Brute-
forcing steghide password (this may take time)...")
```
Explanation: Defines `brute_force_steghide()` to attempt a brute-force attack on a Steghideprotected file using a wordlist.
python
```
try: with open(wordlist_path, "r", encoding="latin-1") as f: passwords =
f.read().splitlines()
```
Explanation: Opens the wordlist file, reading its contents line by line, storing passwords in a list.
python
```
for pw in passwords:
result = subprocess.run( ["steghide", "extract", "-sf", file_path, "-p",
pw, "-xf", "extracted_secret.txt"], capture_output=True, text=True )
```
Explanation: Iterates through the passwords, trying each one with `steghide extract`. If successful, it extracts a hidden file (`extracted_secret.txt`).
python

```
if "wrote extracted data to" in result.stdout.lower(): print(f"[+] Password
found: {pw}")
print("[+] Extracted file: extracted_secret.txt") return
```

Explanation: Checks if the output indicates successful extraction. If a correct password is found, it prints the password and exits the function.
python

```
print("[-] Password not found in wordlist.")
```

Explanation: If no valid password is found, prints a failure message. python

```
except Exception as e:
print(f"[-] Error in brute-force: {e}")
```

Explanation: Catches errors related to file access or command execution. python

```
def main(): file_path = input("Enter path to stego file (e.g.,
/home/user/stego.jpg): ").strip()
```

Explanation: Defines `main()`, prompts the user for a file path, and removes extra whitespace. python

```
if not os.path.isfile(file_path): print("[-] File not found.") return
```

Explanation: Checks if the provided file exists. If not, prints an error message and exits.
python

```
run_exiftool(file_path) run_strings(file_path) run_binwalk(file_path)
run_foremost(file_path)
```

Explanation: Calls all previously defined analysis functions. python

```
choice = input("\nDo you want to brute-force steghide password? (y/n):
").strip().lower()
```

Explanation: Asks the user if they want to perform brute-force password cracking. python

```
if choice == 'y':
wordlist_path = input("Enter path to wordlist (e.g.,
/usr/share/wordlists/rockyou.txt): ").strip() if
os.path.isfile(wordlist_path): brute_force_steghide(file_path,
wordlist_path) else: print("[-] Wordlist file not found.")
```

Explanation: If the user selects brute-force, asks for a wordlist file. If the file exists, runs `brute_force_steghide()`.
python

```
if __name__ == "__main__": main()
```

Explanation: Ensures the script runs only when executed directly, not when imported as a module.