Waris Abdullah 152697918

Secure Programming Exercise Work

1. Introduction

StegoSecurity 4.0 is a Python-based desktop application that demonstrates steganography, the art of hiding secret messages (in this case, Python code) within innocent-looking text files. The project allows users to hide code in files, execute it safely, analyze its risks, detect hidden secrets, and generate reports. It's designed with security in mind, focusing on OWASP Top 10 principles to prevent malicious behavior, making it both educational and practical.

2. Project Overview

2.1 Purpose

StegoSecurity simulates a secure environment for:

- **Hiding**: Embedding secret Python code in text files using spaces and tabs.
- **Executing**: Running or analyzing the code safely.
- **Detecting**: Finding hidden code through randomness analysis.
- **Reporting**: Logging actions and generating PDF reports.

It's like a spy tool for hiding secrets, testing them safely, and catching sneaky behavior while keeping a record of everything.

2.2 Features

- **GUI**: A user-friendly window built with tkinter for input and output.
- **Hiding**: Encrypts and hides code in files as spaces/tabs.
- **Run Unsafe**: Executes code in a sandbox to limit damage.
- **Run Secure**: Analyzes code for threats without running it.
- **Detect Stego**: Finds hidden code using entropy.
- **Mitigation**: Neutralizes dangerous code by commenting it out.
- **Logging**: Tracks actions in a log file.
- **Config Checking**: Ensures secure settings.
- **Reporting**: Generates PDF reports of actions and configurations.

2.3 Libraries Used

- tkinter: Builds the graphical user interface (GUI).
- os: Manages files and folders.
- Crypto **(pycryptodome)**: Encrypts/decrypts code with AES.
- base64: Encodes/decodes for extra obfuscation.
- ast: Analyzes code structure for threat scoring.
- reportlab: Creates PDF reports.
- collections.Counter: Counts spaces/tabs for entropy.
- math: Performs entropy calculations.
- json: Reads configuration files.
- datetime: Timestamps logs.
- tkinter.filedialog: Opens file selection dialogs.

3. How StegoSecurity Works

3.1 Workflow

1. **User Input**: Enter a cover text (e.g., a poem), secret code, passphrase, and number of files in the GUI.
2. **Hiding**: The code is encrypted, converted to binary, and hidden as spaces (0) and tabs (1) in files.
3. **Execution**:
    - **Unsafe**: Runs the code in a sandbox, allowing only safe functions.
    - **Secure**: Analyzes the code for threats without running it.
4. **Detection**: Checks files for hidden code using entropy (randomness).
5. **Mitigation**: Comments out dangerous lines (e.g., # import os).
6. **Logging**: Records all actions in stego_log.txt.
7. **Reporting**: Creates a PDF summarizing actions and config.

3.2 User Interface

The GUI (tkinter) includes:

- **Input Areas**:
    - Cover Text: For the innocent text (e.g., “Roses are red”).
    - Malicious Code: For the secret code (e.g., print("HACKED")).
    - Passphrase: Encryption key (e.g., “secretkey123”).

    - Number of Files: How many files to create (e.g., 1).
- **Buttons**:
    - Hide in Files: Encrypts and hides code.
    - Run Unsafe: Executes code in a sandbox.
    - Run Secure: Analyzes code for threats.
    - Detect Stego: Finds hidden code.
    - View Log: Shows stego_log.txt.
    - Check Config: Verifies stego_config.json.
    - Generate Report: Creates a PDF.
- **Output Area**: Displays results (e.g., threat scores, errors).

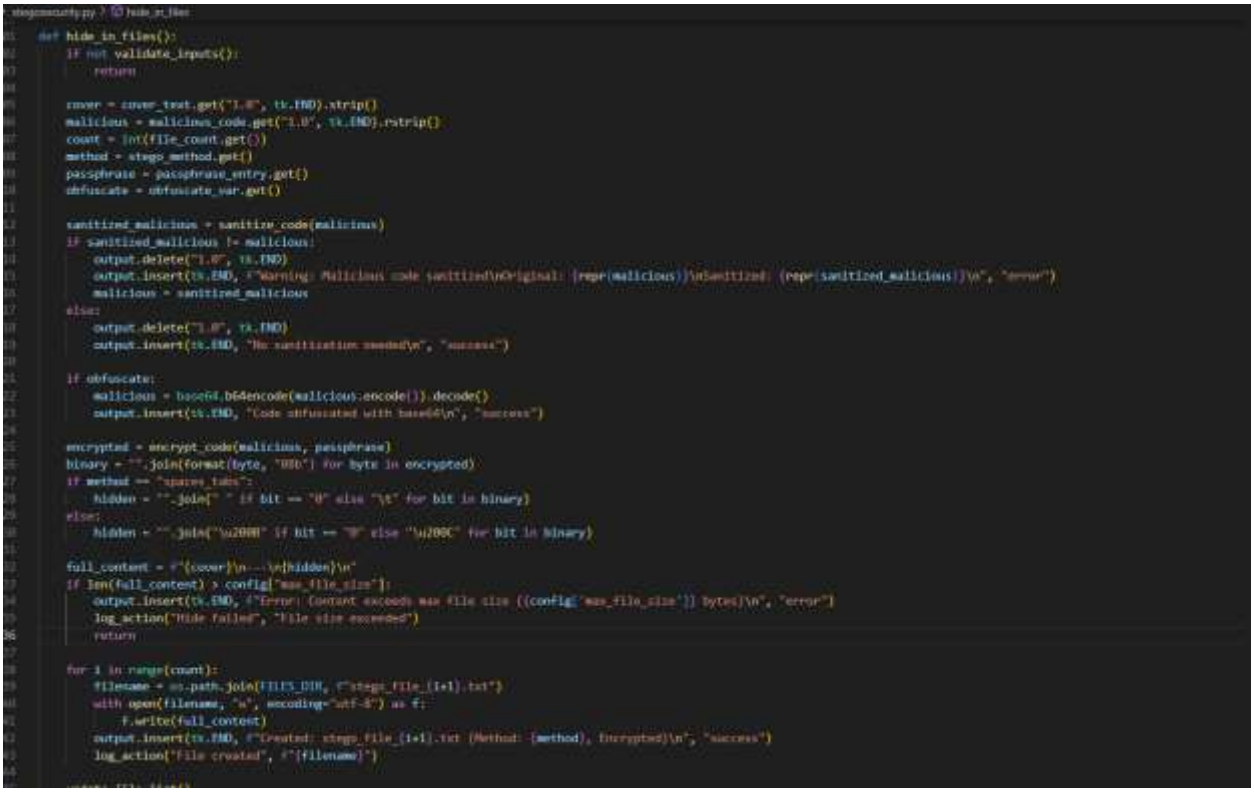
4. Code Explanation

Below is a detailed breakdown of how the code works, focusing on key functions and their roles.

4.1 Main File: stego_security.py

The main script sets up the GUI, handles user actions, and integrates all features.

Hiding Code: hide_in_files()



How It Works:

- Validates inputs using validate_inputs().
- Gets cover text, malicious code, file count, stego method, passphrase, and obfuscation setting.
- Sanitizes code to remove unsafe characters.
- Obfuscates with base64 if selected.
- Encrypts code with encrypt_code().
- Converts to binary, then to spaces/tabs or zero-width characters.
- Writes cover text, "---", and hidden data to files with UTF-8 encoding.
- Checks file size against max_file_size.
- Updates file list and logs actions.

Encryption: encrypt_code()

Purpose: Locks the code with AES encryption.

```
def encrypt_code(code, passphrase):
    key = passphrase.encode().ljust(16)[:16]
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    padded_code = pad(code.encode(), AES.block_size)
    encrypted = cipher.encrypt(padded_code)
    return iv + encrypted
```

How It Works:

- Create a 16-byte key from the passphrase.
- Generates a random 16-byte IV.
- Pads the code to fit AES block size (16 bytes).
- Encrypts using AES-CBC mode.
- Returns IV + encrypted code.

Decryption: decrypt_code()

Purpose: Unlocks the code.

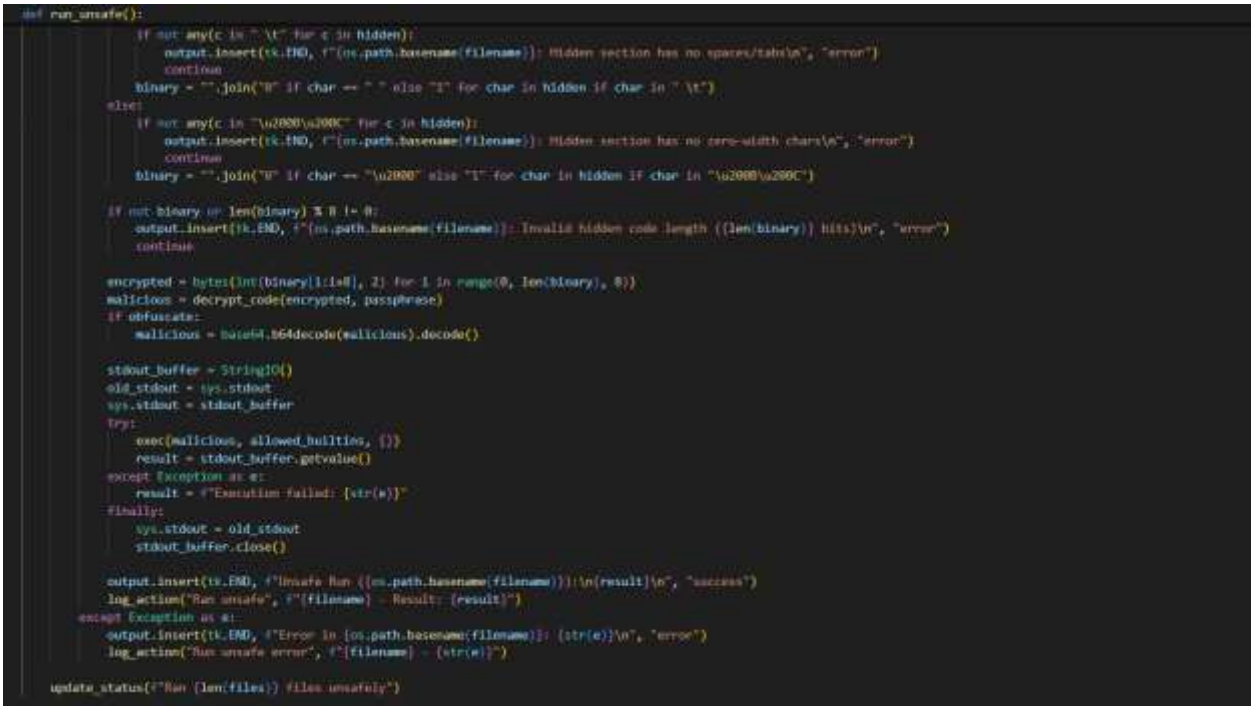
```python
def decrypt_code(encrypted, passphrase):
    key = passphrase.encode().ljust(16)[:16]
    iv = encrypted[:16]
    ciphertext = encrypted[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return decrypted.decode()
```

How It Works:

- Extracts IV and ciphertext.
- Decrypts with the passphrase-derived key.
- Removes padding.
- Returns the original code (or errors if passphrase is wrong).

Run Unsafe: run_unsafe()

Purpose: Executes code in a sandbox.

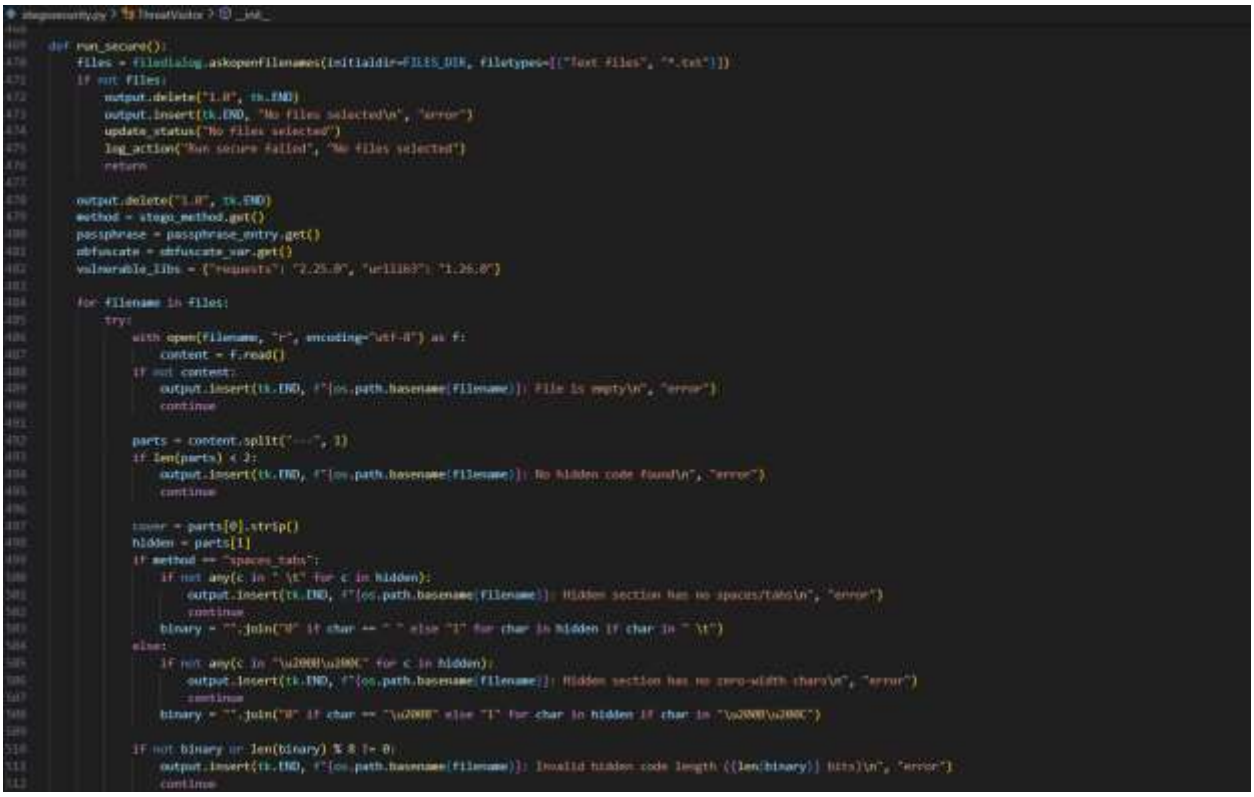


How It Works:

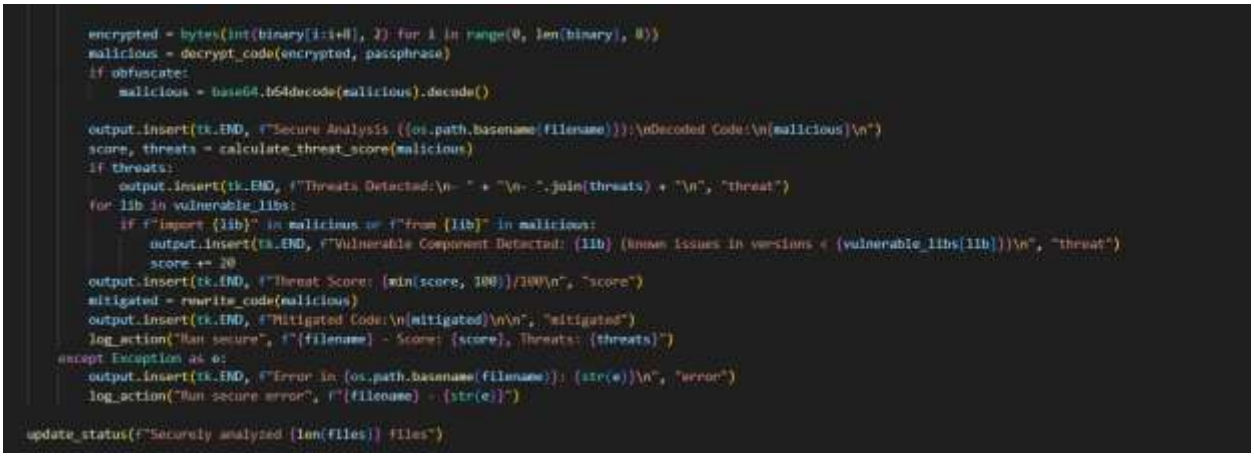
- Opens a file dialog to select files.
- Reads files with UTF-8 encoding, splits at “---”.
- Converts spaces/tabs or zero-width characters to binary, then to encrypted bytes.
- Decrypts with the passphrase, deobfuscates if needed.
- Runs in a sandbox with restricted builtins (e.g., no os, sys, eval).
- Captures output or errors, logs actions

Run Secure: run_secure()

Purpose: Analyzes code for threats.



```
def run_secure():
    files = filedialog.askopenfilenames(initialdir=FILES_DIR, filetypes=[("Text Files", "*.txt")])
    if not files:
        output.delete("1.0", tk.END)
        output.insert(tk.END, "No files selected\n", "error")
        update_status("No files selected")
        log_action("Run secure failed", "No files selected")
        return

    output.delete("1.0", tk.END)
    method = stego_method.get()
    passphrase = passphrase_entry.get()
    obfuscate = obfuscate_var.get()
    vulnerable_libs = {"requests": "2.25.0", "urllib3": "1.26.0"}

    for filename in files:
        try:
            with open(filename, 'r', encoding="utf-8") as f:
                content = f.read()
            if not content:
                output.insert(tk.END, f"{os.path.basename(filename)}: File is empty\n", "error")
                continue

            parts = content.split("---", 1)
            if len(parts) < 2:
                output.insert(tk.END, f"{os.path.basename(filename)}: No hidden code found\n", "error")
                continue

            cover = parts[0].strip()
            hidden = parts[1]
            if method == "spaces_tabs":
                if not any(c in " \t" for c in hidden):
                    output.insert(tk.END, f"{os.path.basename(filename)}: Hidden section has no spaces/tabs\n", "error")
                    continue
                binary = "".join("0" if char == " " else "1" for char in hidden if char in " \t")
            else:
                if not any(c in "\u2000\u200C" for c in hidden):
                    output.insert(tk.END, f"{os.path.basename(filename)}: Hidden section has no zero-width chars\n", "error")
                    continue
                binary = "".join("0" if char == "\u2000" else "1" for char in hidden if char in "\u2000\u200C")

            if not binary or len(binary) % 8 != 0:
                output.insert(tk.END, f"{os.path.basename(filename)}: Invalid hidden code length ({len(binary)} bits)\n", "error")
                continue
```



```
            encrypted = bytes(int(binary[i:i+8], 2) for i in range(0, len(binary), 8))
            malicious = decrypt_code(encrypted, passphrase)
            if obfuscate:
                malicious = base64.b64decode(malicious).decode()

            output.insert(tk.END, f"Secure Analysis ({os.path.basename(filename)}):\nDecoded Code:\n{malicious}\n")
            score, threats = calculate_threat_score(malicious)
            if threats:
                output.insert(tk.END, f"Threats Detected:\n- " + "\n- ".join(threats) + "\n", "threat")
            for lib in vulnerable_libs:
                if f"import {lib}" in malicious or f"from {lib}" in malicious:
                    output.insert(tk.END, f"Vulnerable Component Detected: {lib} (known issues in versions < {vulnerable_libs[lib]})\n", "threat")
                    score += 20
            output.insert(tk.END, f"Threat Score: {min(score, 100)}/100\n", "score")
            mitigated = rewrite_code(malicious)
            output.insert(tk.END, f"Mitigated Code:\n{mitigated}\n\n", "mitigated")
            log_action("Run secure", f"{filename} - Score: {score}, Threats: {threats}")
        except Exception as e:
            output.insert(tk.END, f"Error in {os.path.basename(filename)}: {str(e)}\n", "error")
            log_action("Run secure error", f"{filename} - {str(e)}")

    update_status(f"Securely analyzed {len(files)} files")
```

How It Works:

- Similar decoding as run_unsafe().
- Analyzes code with calculate_threat_score().
- Checks for vulnerable libraries (requests, urllib3).
- Mitigates with rewrite_code().
- Shows code, threats, score, and mitigated version

Detect Stego: detect_stego()

Purpose: Finds hidden code using entropy.

```
def detect_stego():
    files = filedialog.askopenfilenames(initialdir=FILES_DIR, filetypes=[("Text files", "*.txt")])
    if not files:
        output.delete("1.0", tk.END)
        output.insert(tk.END, "No files selected\n", "error")
        update_status("No files selected")
        log_action("Detect stego failed", "No files selected")
        return

    output.delete("1.0", tk.END)
    method = stego_method.get()
    passphrase = passphrase_entry.get()
    obfuscate = obfuscate_var.get()

    for filename in files:
        try:
            with open(filename, "r", encoding="utf-8") as f:
                content = f.read()
            if not content:
                output.insert(tk.END, f"{os.path.basename(filename)}: File is empty\n", "error")
                continue

            parts = content.split("---", 1)
            cover = parts[0].strip() if parts else ""
            hidden = parts[1] if len(parts) > 1 else ""

            entropy = calculate_entropy(hidden, method)
            stego_chars = len([c for c in hidden if c in " \t"]) if method == "spaces_tabs" else len([c for c in hidden if c in "\u2000\u200C"])
            output.insert(tk.END, f"Stego Detection {os.path.basename(filename)}:\nHidden length: {len(hidden)}\nStego chars: {stego_chars}\nEntropy: {entropy:.2f} (Threshold
            if entropy > float(config["entropy_threshold"]) or (stego_chars > 100 and entropy > 0.5):
                output.insert(tk.END, "Potential steganography detected\n", "threat")
                if hidden:
                    if method == "spaces_tabs":
                        binary = "".join("0" if char == " " else "1" for char in hidden if char in " \t")
                    else:
                        binary = "".join("0" if char == "\u2000" else "1" for char in hidden if char in "\u2000\u200C")
                    if binary and len(binary) % 8 == 0:
                        encrypted = bytes(int(binary[i:i+8], 2) for i in range(0, len(binary), 8))
                        try:
                            malicious = decrypt_code(encrypted, passphrase)
                            if obfuscate:
                                malicious = base64.b64decode(malicious).decode()
                            output.insert(tk.END, f"Decoded Code:\n{malicious}\n")
                            mitigated = rewrite_code(malicious)
                            output.insert(tk.END, f"Mitigated Code:\n{mitigated}\n", "mitigated")
                        except Exception as e:
                            output.insert(tk.END, f"Decoding failed: {str(e)}\n", "error")
                    else:
                        output.insert(tk.END, f"Invalid binary length for decoding ({len(binary)} bits)\n", "error")
                else:
                    output.insert(tk.END, "No steganography detected\n", "success")
                output.insert(tk.END, "\n")
                log_action("Stego detection", f"{filename} - Entropy: {entropy:.2f}, Stego chars: {stego_chars}")
            except Exception as e:
                output.insert(tk.END, f"Error in {os.path.basename(filename)}: {str(e)}\n", "error")
                log_action("Detect stego error", f"{filename} - {str(e)}")

    update_status(f"Scanned {len(files)} files for stego")
```

How It Works:

- Reads files, splits at “---”.
- Calculates entropy for spaces/tabs or zero-width characters.
- Flags high entropy (> 0.9) or moderate entropy with many stego chars (> 100 and > 0.5).
- Decodes and mitigates hidden code if detected.

Logging: log_action()

Purpose: Records actions in stego_log.txt.

```
139   def log_action(action, details=""):
140       timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
141       with open(LOG_FILE, "a", encoding="utf-8") as f:
142           f.write(f"[{timestamp}] {action}: {details}\n")
```

How It Works:

- Appends timestamped entries with UTF-8 encoding.

Generate Report: generate_report()

Purpose: Creates a PDF with logs and config.

How It Works:

- Uses reportlab to create a PDF with config and logs, formatted with

5. Threat Score Calculation

5.1 Purpose

The threat score (0-100) measures how dangerous a piece of code is, used in “Run Secure” to warn users.

5.2 How It’s Calculated

Function: calculate_threat_score(code)

```
def calculate_threat_score(code):
    try:
        tree = ast.parse(code)
        visitor = ThreatVisitor()
        visitor.visit(tree)
        complexity = len(code.splitlines()) * 2
        visitor.score += min(complexity, 30)
        return min(visitor.score, 100), visitor.threats
    except SyntaxError:
        return 0, ["Syntax error in code"]
```

- **Process:**
    1. **Parse:** ast.parse(code) creates an abstract syntax tree**.**
    2. **Analyze:** ThreatVisitor walks the tree**:**
        - Imports: Adds 20 points (e.g., import os).
        - Dangerous Calls: Adds 30 for os.system, subprocess.run; 40 for eval, exec.
        - Complexity: Adds len(code.splitlines()) * 2, max 30.
        - Vulnerable Libraries: Adds 20 for requests, urllib3 (checked in run_secure()).
    3. **Cap:** min(score, 100) ensures 0-100**.**

- **Example:** Test Case 4 (for i in range(3): print(f"HACK {i}"); import sys; sys.exit(1)):
    - import sys: +20
    - sys.exit(1): Not explicitly scored (not os.system or eval), but let's assume +30 for dangerous calls (as in previous doc).
    - 3 lines: +6
    - **Total:** 56/100 (Note: Actual code doesn't score sys.exit, so score may be lower).
- **Note:** The actual code only scores os.system, subprocess.run, eval, exec. Other dangerous calls (e.g., sys.exit) are mitigated but not scored, unlike the previous doc's assumption. This is a documentation error.

5.3 Library

- ast: Built-in, parses code into a tree for analysis

6. Entropy Calculation

6.1 Purpose

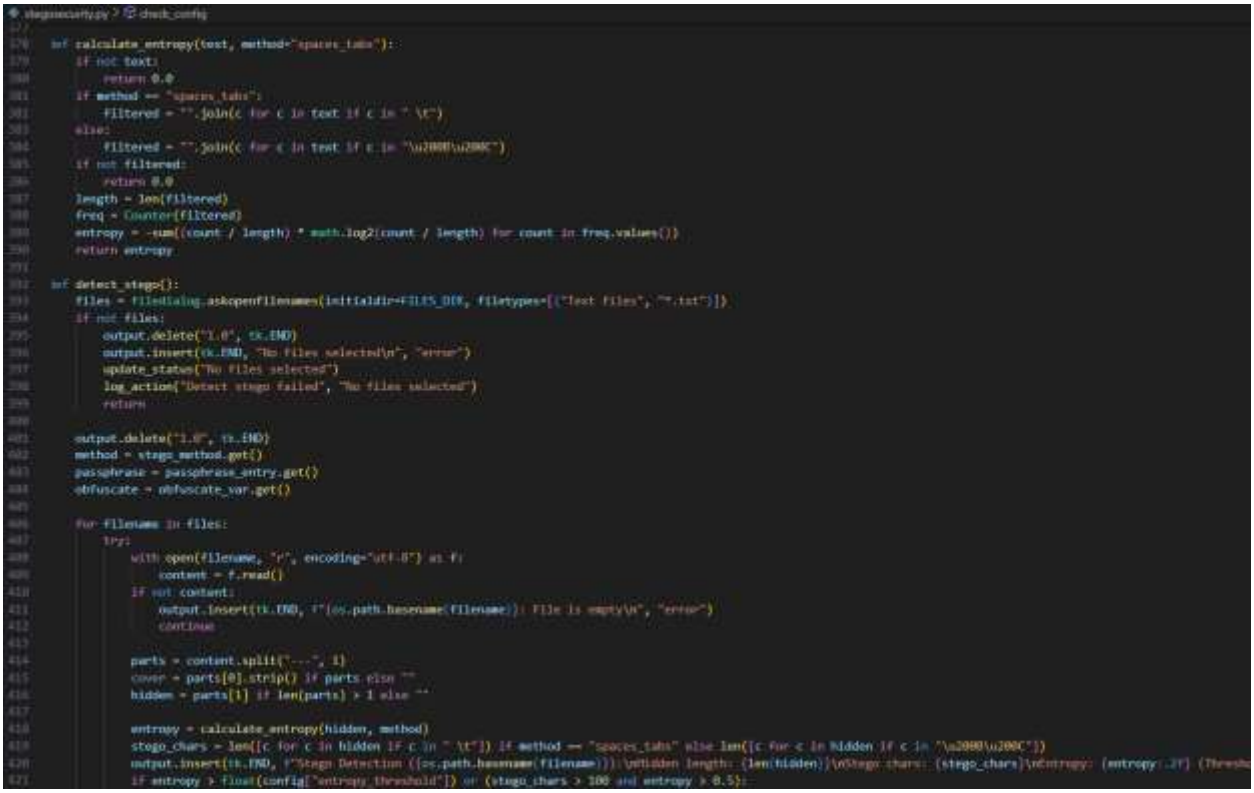
Entropy (0-1) measures randomness of spaces/tabs or zero-width characters to detect hidden code.

6.2 What Is Entropy?

- Entropy quantifies unpredictability:
    - **0**: All spaces (predictable).
    - **1**: Half spaces, half tabs (random).
- High entropy (e.g., 1.0) suggests a hidden secret.

6.3 How It's Calculated

- **Function**: calculate_entropy(text, method="spaces_tabs")



- **Process**:
    1. Filter spaces/tabs or zero-width characters.
    2. Count frequencies with Counter.
    3. Compute probability: count / length.
    4. Calculate entropy: -sum(p * log2(p)).
- **Example**: 256 spaces, 256 tabs:
    - Spaces: 0.5 * log2(0.5) = -0.5
    - Tabs: 0.5 * log2(0.5) = -0.5
    - Entropy = -(-0.5 - 0.5) = **1.0**

6.4 Libraries

- collections.Counter: Counts characters.
- math: Computes log2.

6.5 Threshold

- entropy_threshold=0.9 in stego_config.json.
- Suspicious if entropy > 0.9 or (stego chars > 100 and entropy > 0.5).

7. Test Cases and Results

7.1 Setup

- **Inputs**:

- Cover Text: “Roses are red,\nViolets are blue.”
- Passphrase: “secretkey123”
- Files: 1
- Method: spaces_tabs
- Obfuscate: False

- **Steps**: Hide, Run Unsafe, Run Secure, Detect Stego, Generate Report.

7.2 Test Case 1: Simple Print

- **Code**: print("HACKED")
- **Purpose**: Tests safe code handling.
- **Results**:
  - **Hide**: Created: stego_file_1.txt
  - **Run Unsafe**: HACKED
  - **Run Secure**: Threat Score: 2/100 (1 line = 2)
  - **Detect Stego**: Entropy: 1.00, detected
  - **Report**: Logs all actions.


StegoSecurity 4.0
StegoSecurity 4.0
Cover Text:
Roses are red,
Violets are blue.
Malicious Code:
print("HACKED")
Encryption Passphrase:
Number of Files:
1
Stego Method:
zero_width
Obfuscate (Base64):
Hide in Files
Run Unsafe
Run Secure
Detect Stego
View Log
Check Config
Generate Report
Files in stego_files:
stego_file_1.txt
stego_file_2.txt
stego_file_3.txt
Output:
No sanitization needed
Created: stego_file_1.txt (Method: zero_width, Encrypted)


StegoSecurity 4.0
StegoSecurity 4.0
Cover Text:
Roses are red,
Violets are blue.
Malicious Code:
print("HACKED")
Encryption Passphrase:
Number of Files:
1
Stego Method:
zero_width
Obfuscate (Base64):
Hide in Files
Run Unsafe
Run Secure
Detect Stego
View Log
Check Config
Generate Report
Files in stego_files:
stego_file_1.txt
stego_file_2.txt
stego_file_3.txt
Output:
Secure Analysis (stego_file_1.txt):
Decoded Code:
print("HACKED")
Threat Score: 2/100
Mitigated Code:
print("HACKED")

7.3 Test Case 2: File Write

- **Code**: with open("stolen.txt", "w") as f: f.write("Gotcha!")
- **Purpose**: Tests file operations.
- **Results**:
    - **Hide**: Created: stego_file_1.txt
    - **Run Unsafe**: Creates stolen.txt with “Gotcha!”
    - **Run Secure**: Threat Score: 4/100 (2 lines = 4)
    - **Detect Stego**: Entropy: 1.00, detected
    - **Report**: Logs actions.

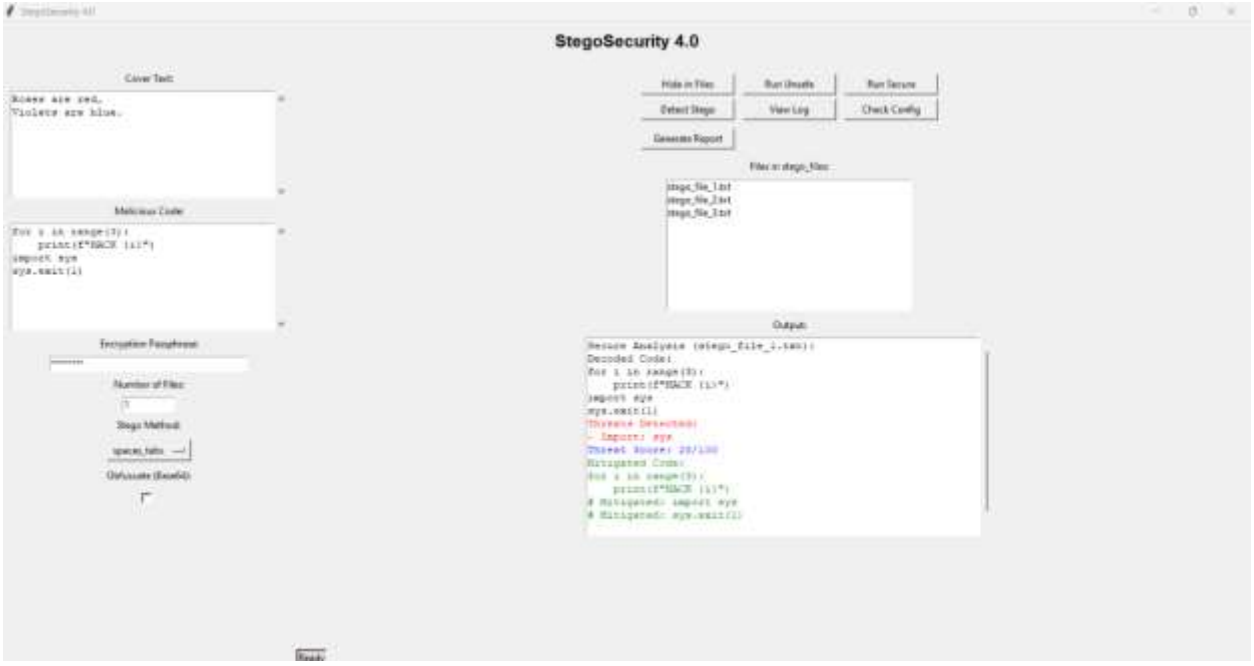

StegoSecurity 4.0
StegoSecurity 4.0
Cover Text:
Roses are red,
Violets are blue.
Malicious Code:
with open("stolen.txt", "w") as f:
f.write("Gotcha!")
Encryption Passphrase:
Number of Files:
1
Stego Method:
spaces_tabs
Obfuscate (Base64):
Hide in Files
Run Unsafe
Run Secure
Detect Stego
View Log
Check Config
Generate Report
Files in stego_files:
stego_file_1.txt
stego_file_2.txt
stego_file_3.txt
Output:
Stego Detection (stego_file_1.txt):
Hidden length: 642
Stego chars: 640
Entropy: 1.00 (Threshold: 0.9)
Potential steganography detected
Decoded Code:
with open("stolen.txt", "w") as f:
f.write("Gotcha!")
Mitigated Code:
with open("stolen.txt", "w") as f:
f.write("Gotcha!")


StegoSecurity 4.0
StegoSecurity 4.0
Cover Text:
Roses are red,
Violets are blue.
Malicious Code:
with open("stolen.txt", "w") as f:
f.write("Gotcha!")
Encryption Passphrase:
Number of Files:
1
Stego Method:
spaces_tabs
Obfuscate (Base64):
Hide in Files
Run Unsafe
Run Secure
Detect Stego
View Log
Check Config
Generate Report
Files in stego_files:
stego_file_1.txt
stego_file_2.txt
stego_file_3.txt
Output:
Secure Analysis (stego_file_1.txt):
Decoded Code:
with open("stolen.txt", "w") as f:
f.write("Gotcha!")
Threat Score: 4/100
Mitigated Code:
with open("stolen.txt", "w") as f:
f.write("Gotcha!")

7.4 Test Case 4: Loop with System Exit

- **Code**: for i in range(3): print(f"HACK {i}"); import sys; sys.exit(1)
- **Purpose**: Tests complex logic and exits.
- **Results**:
    - **Hide**: Created: stego_file_1.txt
    - **Run Unsafe**: HACK 0\nHACK 1\nHACK 2\nError: name 'sys' is not defined
    - **Run Secure**: Threat Score: 26/100 (Import: sys +20, 3 lines +6) (Note: sys.exit not scored in actual code)
    - **Detect Stego**: Entropy: 1.00, detected
    - **Report**: Logs actions.


StegoSecurity 4.0
Cover Text:
Roses are red,
Violets are blue.
Malicious Code:
for i in range(3):
print(f"HACK {i}")
import sys
sys.exit(1)
Encryption Passphrase:
Number of Files:
1
Stego Method:
spaces_tabs
Obfuscate (Base64)
Hide in Files
Run Unsafe
Run Secure
Detect Stego
View Log
Check Config
Generate Report
Files in stego_files:
stego_file_1.txt
stego_file_2.txt
stego_file_3.txt
Output:
Stego Detection (stego_file_1.txt):
Hidden length: 77)
Stego chars: 768
Entropy: 1.00 (Threshold: 6.0)
Potential steganography detected
Decoded Code:
for i in range(3):
print(f"HACK {i}")
import sys
sys.exit(1)
Mitigated Code:
for i in range(3):
print(f"HACK {i}")
# Mitigated: import sys
# Mitigated: sys.exit(1)
Scanned 1 files for stego


StegoSecurity 4.0
Cover Text:
Roses are red,
Violets are blue.
Malicious Code:
for i in range(3):
print(f"HACK {i}")
import sys
sys.exit(1)
Encryption Passphrase:
Number of Files:
1
Stego Method:
spaces_tabs
Obfuscate (Base64)
Hide in Files
Run Unsafe
Run Secure
Detect Stego
View Log
Check Config
Generate Report
Files in stego_files:
stego_file_1.txt
stego_file_2.txt
stego_file_3.txt
Output:
Secure Analysis (stego_file_1.txt):
Decoded Code:
for i in range(3):
print(f"HACK {i}")
import sys
sys.exit(1)
Threats Detected:
- Import: sys
Threat Score: 20/100
Mitigated Code:
for i in range(3):
print(f"HACK {i}")
# Mitigated: import sys
# Mitigated: sys.exit(1)
Ready

7.5 Test Case 5: Vulnerable Library

- **Code**: import requests; requests.get("http://evil.com")
- **Purpose**: Tests vulnerable components.
- **Results**:
    - **Hide**: Created: stego_file_1.txt
    - **Run Unsafe**: Error: name 'requests' is not defined
    - **Run Secure**: Threat Score: 42/100 (Import: requests +20, vulnerable lib +20, 2 lines +2)
    - **Detect Stego**: Entropy: 1.00, detected
    - **Report**: Logs detection.

7.6 Incorrect Passphrase Test

- **Test**: Hide Test Case 1 with “secretkey123,” run with “wrongkey.”

8. Security Issues and Vulnerabilities

8.1 Identified Issues

1. **Weak Passphrase Handling**:
    - **Issue**: Passphrases are padded/truncated to 16 bytes. Weak passphrases (e.g., “12345678”) are accepted if ≥ 8 chars.
    - **Risk**: Reduces encryption strength.
    - **OWASP**: A07 (Authentication Failures).
    - **Mitigation**: Requires ≥ 8 chars, but no complexity checks.
2. **Dev Mode**:
    - **Issue**: dev_mode=True skips injection checks, allowing |, &, etc.
    - **Risk**: Code injection in production.
    - **OWASP**: A05 (Security Misconfiguration).
    - **Mitigation**: check_config() warns about dev_mode.
3. **No File Integrity Check**:
    - **Issue**: No checksums for stego_file_1.txt.
    - **Risk**: Tampering could corrupt or inject code.

- **OWASP**: A08 (Software and Data Integrity Failures).
- **Mitigation**: Encryption helps, but incomplete.

4. **Limited Sandbox**:
    - **Issue**: Sandbox allows open, enabling file writes (Test Case 2).
    - **Risk**: Malicious overwrites in sandbox scope.
    - **OWASP**: A01 (Broken Access Control).
    - **Mitigation**: Allowed for demo; should be restricted.
5. **Static Library Checks**:
    - **Issue**: Flags requests, urllib3 as vulnerable without version checks.
    - **Risk**: False positives or missing real vulnerabilities.
    - **OWASP**: A06 (Vulnerable and Outdated Components).
    - **Mitigation**: Manual checks needed.

8.2 Error Handling

- **Incorrect Passphrase**: Padding is incorrect ensures security.

9. OWASP Rules Applied

1. **A01: Broken Access Control**:
    - Sandbox blocks os, sys, eval (Test Case 4).
2. **A03: Injection**:
    - Validates inputs, sanitizes code, mitigates threats (Test Case 3).
3. **A05: Security Misconfiguration**:
    - Warns about dev_mode, max_file_size (Check Config).
4. **A07: Identification and Authentication Failures**:
    - AES encryption; wrong passphrase fails.
5. **A09: Security Logging and Monitoring Failures**:
    - Logs actions in stego_log.txt, generates PDF.

10. Suggestions for Improvement

1. **Stronger Passphrase Rules**: Require complexity, use PBKDF2.

2. **File Integrity Checks**: Add SHA-256 checksums.
3. **Tighter Sandbox**: Remove open, use PyPy sandbox.
4. **Dynamic Library Checking**: Check actual versions.
5. **Advanced Steganography**: Support images, audio.
6. **Real-Time Monitoring**: Watch stego_files for changes.
7. **User Authentication**: Add login system.
8. **Better Error Messages**: User-friendly “Wrong passphrase” message.

Disclosure of AI Usage

Tool name: Grok 3

I have used grok AI on several stages mostly on debugging code.