**Project Outline: Hosts-Based Domain Blocker**

**Part 1: Group Formation & Case Study Selection**

1. **Group Formation:**
   * Form a group of 2-3 members.
   * Ensure effective communication and collaboration.
   * Use collaborative tools like Slack or Microsoft Teams for communication and Trello for task management.
2. **Case Study Selection:**
   * Choose the "Hosts-Based Domain Blocker" case study.
   * Understand the requirements: blocking access to known ad and malware domains by updating the hosts file.
3. **Initial Meeting & Brainstorming:**
   * Conduct a brainstorming session to discuss the project scope.
   * Choose Python for its cross-platform capabilities and extensive libraries for network operations.
   * Identify current process requirements and the benefits of automating the domain blocking.
   * Set organizational guidelines: follow best practices in coding, testing, and documentation.

**Part 2: Design the Software Solution**

1. **Team Meetings:**
   * Hold regular meetings to brainstorm and refine ideas.
   * Document discussions with meeting minutes and notes.
2. **Algorithm Design:**
   * Design an algorithm using sequence, selection, and iteration constructs.
   * Ensure the algorithm handles all user options and scenarios.
   * Create a flowchart to represent the algorithm visually.
   * Write pseudocode to outline the program logic.
   * Use UML diagrams to illustrate the system architecture and component interactions.
3. **Flowchart Example:**

Start

|

v

Read configuration file.

|

v

Display content block options.

|

v

User selects content type to block.

|

v

Download and parse domain block lists.

|

v

Merge lists into unique domains.

|

v

Backup original hosts file.

|

v

Append blocked domains to hosts file.

|

V

End

1. **Pseudocode Example:**

def read\_config(config\_file):

# Read URLs from the configuration file

pass

def download\_block\_lists(urls):

# Download domain block lists from URLs

pass

def parse\_block\_lists(lists):

# Parse and merge lists into unique domains

pass

def backup\_hosts\_file():

# Backup the original hosts file

pass

def append\_to\_hosts\_file(domains):

# Append blocked domains to the hosts file

pass

def main():

config = read\_config("config.ini")

urls = config["urls"]

lists = download\_block\_lists(urls)

domains = parse\_block\_lists(lists)

backup\_hosts\_file()

append\_to\_hosts\_file(domains)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Part 3: Implement the Solution**

1. **Implementation:**
   * Translate the algorithm into Python code.
   * Ensure the program is user-friendly and handles all scenarios.
2. **Testing:**
   * Perform thorough testing to ensure the program works as expected.
   * Correct any errors or unhandled situations.
   * Get feedback from peers or other groups to identify potential issues.
3. **Version Control:**
   * Use GitHub for collaborative coding and version control.
   * Maintain a clear commit history and use branches for feature development.

**Part 4: Create Documentation**

1. **Developer-Level Documentation:**
   * Document the purpose, architecture, and flow of the program.
   * Include detailed comments within the source code.
   * Explain complex or critical sections of the code for clarity.
2. **User-Level Documentation:**
   * Create a user manual with step-by-step instructions for using the program.
   * Include troubleshooting tips and FAQs.

**Part 5: Compile the Final Report**

1. **Report Compilation:**
   * Compile all work into a final report using the provided templates.
   * Ensure the report is clear, professional, and well-organized.
2. **Final Report Contents:**
   * Initial case study analysis and brainstorming session notes.
   * Algorithm design representations (flowcharts, pseudocode, UML diagrams).
   * Detailed implementation documentation.
   * Testing and debugging records.
   * Developer and user-level documentation.
   * Records of team meetings and discussions.

**Part 6: Final Submission**

1. **Submission:**
   * Each group member submits a copy of the final report, source code files, and user manuals.
   * Ensure all documents are correctly formatted and complete.

**Conclusion**

This project will engage your team in every phase of the software development lifecycle, from initial analysis to final documentation. By automating the process of updating the hosts file to block malicious domains, you will enhance network security and gain valuable experience in software development and team collaboration.

Chatgpt’s code

import configparser

import os

import requests

def read\_config(config\_file):

# Read URLs from the configuration file

config = configparser.ConfigParser()

config.read(config\_file)

return config['DEFAULT']

def download\_block\_lists(urls):

# Download domain block lists from URLs

lists = []

for url in urls:

response = requests.get(url)

if response.status\_code == 200:

lists.append(response.text.splitlines())

else:

print(f"Failed to download block list from {url}")

return lists

def parse\_block\_lists(lists):

# Parse and merge lists into unique domains

domains = set()

for block\_list in lists:

for domain in block\_list:

domains.add(domain.strip())

return domains

def backup\_hosts\_file():

# Backup the original hosts file

hosts\_path = '/etc/hosts' # Adjust path based on your OS

backup\_path = '/etc/hosts.backup' # Adjust path based on your preference

if os.path.exists(hosts\_path) and not os.path.exists(backup\_path):

os.system(f"cp {hosts\_path} {backup\_path}")

def append\_to\_hosts\_file(domains):

# Append blocked domains to the hosts file

hosts\_path = '/etc/hosts' # Adjust path based on your OS

with open(hosts\_path, 'a') as hosts\_file:

for domain in domains:

hosts\_file.write(f"127.0.0.1 {domain}\n")

def main():

config = read\_config("config.ini")

urls = config.get("urls", "").split(",")

lists = download\_block\_lists(urls)

domains = parse\_block\_lists(lists)

backup\_hosts\_file()

append\_to\_hosts\_file(domains)

if \_\_name\_\_ == "\_\_main\_\_":

main()