A logo of a digital egypt pioneers

AI-generated content may be incorrect.

**VULNERABILITY ASSESSMENT AND REMEDIATION PLAN**

**Supervised by: Eng. Shehab Elbatal**

**Team members:**

**🡪Omar Alaaeldin Mohamed**

**🡪Abdelrahman Ayman Khairy**

**🡪Ali Mohamed Ali**

Chapter 1: Overview

1.1 Overview

This tool is designed to streamline the process of analyzing Nessus vulnerability scan reports by extracting Common Vulnerabilities and Exposures (CVEs) and querying an AI model (Groq's Llama3-70b-8192) for detailed remediation guidance tailored to the operating system of the affected host. The application provides a graphical user interface (GUI), allowing users to load reports, extract CVEs, send them to the AI API, view results, and export findings.

1.2 Problems That the Tool Will Fix

Manual Analysis Burden : Security analysts often manually look up each CVE from scan reports to understand its impact and how to fix it.

Time-Consuming Remediation Research : Finding official patches or mitigation steps for each CVE can be tedious and error-prone.

Lack of Contextual Guidance : Generic CVE descriptions do not always provide OS-specific remediation steps.

Need for Rapid Reporting : Analysts require fast extraction and actionable output without switching between multiple tools.

Chapter 2: Design, Implementation, and Testing

2.1 Overview

This chapter discusses the design methodology, implementation strategy, and testing phases involved in developing the Nessus-to-Groq integration tool.

2.2 Software Development Approach

The software was developed using a lightweight Agile approach , focusing on iterative development and rapid prototyping to ensure responsiveness and usability.

2.3 Used Approach

An Agile-inspired Waterfall hybrid model was used:

Initial planning and requirement gathering.

GUI and core logic development in parallel.

Continuous testing during development cycles.

2.4 Justification of the Used Approach

Allows flexibility to refine features during development.

Facilitates early detection of bugs in UI and API integration.

Enables incremental improvements without reworking the entire codebase.

2.5 Phases of the Chosen Model Approach

Requirement Gathering

UI Design & Planning

Parsing Logic Implementation

API Integration

Testing and Debugging

Final Packaging and Documentation

2.6 Why It Is Better Than Other Approaches?

More adaptable than pure Waterfall.

Simpler than full Scrum but still supports iterative improvement.

Suitable for small-scale applications with clear functional goals.

2.7 Alternative Approach

A pure Waterfall model could have been used, which would involve completing one phase before moving to the next. However, this would delay feedback and bug fixing until later stages.

2.10 Changes in Design

Initially, the plan included using OpenAI’s GPT directly, but due to cost constraints, Groq was chosen as a free alternative offering similar capabilities.

2.11 Software Modeling Tools Used

Tkinter – For GUI design and interaction.

XML ElementTree – For parsing Nessus XML files.

Requests Library – For interacting with the Groq API.

Threading Module – For asynchronous API calls without freezing the UI.

2.12 Specifications That Distinguish This Software

Integrates vulnerability scanning with AI-powered remediation suggestions.

Tailors responses to the operating system context.

Provides exportable, readable output.

Lightweight and standalone with no external dependencies beyond Python.

Chapter 3: Results and Discussion

3.1 Overview About This Chapter

This chapter presents the results obtained after implementing the tool, including performance, accuracy, and usability analysis.

3.2 Critical Discussion

The tool successfully parses .nessus files and extracts CVE data accurately. The integration with Groq allows for real-time remediation advice that is both contextual and actionable. However, response time depends heavily on API latency and network stability.

3.3 Skills Acquired from the Project

Working with XML data structures.

Calling REST APIs programmatically.

Building responsive GUI applications with Tkinter.

Implementing threading to maintain UI responsiveness.

Writing effective prompts for large language models.

3.4 Development Goals and Results:

A screenshot of a computer

AI-generated content may be incorrect.

**3.5 Findings and Results**

* Efficiently parses complex Nessus XML reports.
* Generates useful and specific remediation steps per CVE.
* Supports manual chat queries for extended use cases.

**3.6 Project Goals**

* Automate post-scan analysis.
* Reduce manual effort in researching CVE fixes.
* Provide a user-friendly interface for security professionals.

**3.7 Future Work**

* Add support for other AI models (e.g., OpenAI, Anthropic).
* Integrate database storage for CVE history tracking.
* Allow exporting to PDF/Markdown formats.
* Add filtering options (by severity, host, OS).

**3.8 New Area of Investigation**

Exploring integration with ticketing systems (e.g., Jira) for automated vulnerability task creation.

**3.9 Features Not Completed Due to Time Constraints**

* Real-time progress percentage display.
* Saving session states.
* Unit tests for critical functions.

**3.10 Summary of the Chapter**

The project achieved all major objectives. The tool enhances analyst efficiency by bridging the gap between scanning and remediation, though there are opportunities for future enhancements.

**Chapter 4: Project Lifecycle**

**4.1 Design Overview**

The design focused on simplicity and usability. A modular structure was adopted to separate concerns: parsing, API calling, and GUI rendering.

**4.2 Software Development Life Cycle (SDLC)**

**4.2.1 Justification for Agile Approach**

Agile allowed continuous refinement of the tool during development and ensured that issues were caught early.

**4.2.2 Alternative Approach**

A traditional Waterfall model would have required more upfront planning and delayed testing until later stages, potentially increasing bug-fix costs.

**4.4 Implementation**

**A computer screen with text and numbers

AI-generated content may be incorrect.**

* **tkinter : GUI framework used to build the application window and interface.**
* **Submodules (filedialog, scrolledtext, etc.) : Used for specific UI components like file selection and scrollable text areas.**
* **xml.etree.ElementTree : Parses XML files (used to extract data from .nessus reports).**
* **requests : Makes HTTP POST requests to the Groq API.**
* **json : Converts Python objects into JSON format for API requests.**
* **threading : Enables background execution of long-running tasks (like API calls) without freezing the GUI.**
* **Groq API key and model name : Hardcoded credentials for authentication and specifying which LLM to use.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Parses the .nessus file using ElementTree.**
* **Iterates over <ReportHost> tags to get host details.**
* **Extracts OS info from <HostProperties><tag name="operating-system">.**
* **For each vulnerability (<ReportItem>), it collects:**
  + **Plugin Name**
  + **Port**
  + **Description**
  + **CVE(s)**
* **Returns list of tuples containing all extracted CVE data per vulnerability.**
* **Handles exceptions for malformed XML or missing fields.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Sends a formatted prompt with CVE and system context to Groq's LLM.**
* **Asks for detailed remediation steps including:**
  + **Root cause**
  + **Manual verification steps**
  + **Patch instructions**
  + **Official patch links**
* **Uses requests to send the JSON-formatted payload to the Groq API.**
* **Returns the AI’s response or an error message if something goes wrong.**

**A computer screen with text and images

AI-generated content may be incorrect.**

* **Opens a file dialog to select a .nessus file.**
* **Clears previous output in the results box.**
* **Calls extract\_cves\_from\_nessus() and displays the found CVEs in the GUI.**
* **Stores the CVEs in a global variable for later use.**
* **Sets up the progress bar based on the number of CVEs.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **send\_to\_chatgpt\_threaded(): Starts a new thread to avoid freezing the GUI during API calls.**
* **send\_to\_chatgpt():**
  + **Checks if there are any CVEs to process.**
  + **Displays a status message.**
  + **Loops through each CVE and sends it to the Groq API.**
  + **Inserts the result back into the GUI.**
  + **Updates the progress bar after each CVE.**

**A computer screen shot of text

AI-generated content may be incorrect.**

* **Gets the contents of the results box.**
* **Checks if it's empty.**
* **Lets the user choose where to save the file.**
* **Writes the contents to a .txt file with UTF-8 encoding.**
* **Shows a success message after saving.**

**A screen shot of a computer code

AI-generated content may be incorrect.**

* **Copies the currently selected text from the results box.**
* **Clears clipboard before appending new text.**
* **Handles error if no text is selected.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Allows users to type and send custom messages to the AI model.**
* **Displays both the user input and AI response in the GUI.**
* **Runs the API call in a background thread to keep the GUI responsive.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

* **Initializes the main Tkinter window.**
* **Adds buttons, labels, progress bar, and input/output boxes.**
* **Binds keyboard shortcuts (e.g., Ctrl+C).**
* **Starts the GUI event loop with mainloop().**

**4.4.1 User Interface**

The GUI uses Tkinter components:

* Buttons for loading files, sending requests, exporting, and copying text.
* Progress bar to show processing status.
* ScrolledText widget for displaying results.
* Input field for custom chat messages.

**4.4.2 Code**

The code is organized into logical blocks:

* File parsing (**extract\_cves\_from\_nessus**)
* API communication (**ask\_groq\_about\_cve**)
* GUI control (**open\_file**, **send\_to\_chatgpt\_threaded**, etc.)

import tkinter as tk

from tkinter import filedialog,ttk, messagebox,scrolledtext

import xml.etree.ElementTree as ET

import requests

import json

import threading

groq\_api\_key = "gsk\_YtcgbZgRNedRzbjKRPayWGdyb3FYcZbC4MqD4LUX0e7ORMJumBft"

groq\_model = "llama3-70b-8192"

current\_cves = []

def extract\_cves\_from\_nessus(file\_path):

cves = []

try:

tree = ET.parse(file\_path)

root = tree.getroot()

for report\_host in root.iter('ReportHost'):

host\_name = report\_host.attrib.get('name', 'Unknown Host')

os\_info = "Unknown OS"

host\_properties = report\_host.find("HostProperties")

if host\_properties is not None:

for tag in host\_properties.iter("tag"):

if tag.attrib.get("name") == "operating-system":

os\_info = tag.text

for report\_item in report\_host.iter('ReportItem'):

plugin\_name = report\_item.attrib.get('pluginName', 'Unknown')

port = report\_item.attrib.get('port', 'N/A')

cve\_tags = [child.text for child in report\_item if child.tag == "cve"]

description = next((child.text for child in report\_item if child.tag in ("description", "synopsis")), "No description provided.")

solution = next((child.text for child in report\_item if child.tag == "solution"), "No solution provided.")

output = next((child.text for child in report\_item if child.tag == "output"), "No output provided.")

risk\_factor = next((child.text for child in report\_item if child.tag == "risk\_factor"), "Unknown")

cvss\_score = next((child.text for child in report\_item if child.tag == "cvss\_base\_score"), "N/A")

see\_also = "\n".join([child.text for child in report\_item if child.tag == "see\_also"]) or "N/A"

exploit = next((child.text for child in report\_item if child.tag == "exploit\_available"), "Unknown")

for cve in cve\_tags:

cves.append((cve, plugin\_name, port, description, solution, output, host\_name, os\_info, risk\_factor, cvss\_score, see\_also, exploit))

return cves

except Exception as e:

messagebox.showerror("خطأ", f"حدث خطأ أثناء تحليل الملف:\n{e}")

return []

def ask\_groq\_about\_cve(cve, plugin, port, desc, solution, output, host, os\_info, risk, cvss, references, exploit\_available):

try:

headers = {

"Authorization": f"Bearer {groq\_api\_key}",

"Content-Type": "application/json"

}

prompt = (

f"You are a professional cybersecurity expert.\n\n"

f"I scanned a production environment using Nessus and found the following vulnerability:\n\n"

f"- CVE ID: {cve}\n"

f"- Affected Host: {host}\n"

f"- Operating System: {os\_info}\n"

f"- Plugin Name: {plugin}\n"

f"- Port: {port}\n"

f"- Risk Factor: {risk}\n"

f"- CVSS Score: {cvss}\n"

f"- Exploit Available: {exploit\_available}\n"

f"- References: {references}\n"

f"- Description: {desc}\n"

f"- Solution (from Nessus): {solution}\n"

f"- Nessus Output: {output}\n\n"

f"Assume the system is in a production environment and cannot be upgraded to the latest OS version immediately.\n\n"

f"Please respond with:\n"

f"1. A brief explanation of the root cause of this CVE.\n"

f"2. How to manually check if this vulnerability is present on {os\_info}.\n"

f"3. Step-by-step patching or mitigation instructions tailored specifically for {os\_info}.\n"

f"4. Official vendor patch references or security advisories if available.\n\n"

f"Format your response clearly using bullet points, numbered steps, and code blocks for command-line instructions where applicable."

)

payload = {

"model": groq\_model,

"messages": [

{"role": "system", "content": "You are a cybersecurity expert."},

{"role": "user", "content": prompt}

],

"temperature": 0.7

}

response = requests.post("https://api.groq.com/openai/v1/chat/completions", headers=headers, data=json.dumps(payload))

if response.status\_code == 200:

return response.json()["choices"][0]["message"]["content"]

return f"[Error {response.status\_code}]: {response.text}"

except Exception as e:

return f"[Exception]: {e}"

def open\_file():

global current\_cves

file\_path = filedialog.askopenfilename(filetypes=[("Nessus Files", "\*.nessus")])

if not file\_path:

return

current\_cves = extract\_cves\_from\_nessus(file\_path)

for item in tree.get\_children():

tree.delete(item)

for index, cve\_data in enumerate(current\_cves):

tree.insert("", tk.END, iid=index, values=(cve\_data[0], cve\_data[6], cve\_data[7], cve\_data[1], cve\_data[2], cve\_data[8]))

def send\_selected\_to\_groq():

selected\_items = tree.selection()

if not selected\_items:

messagebox.showinfo("تنبيه", "اختر واحدة أو أكثر من الثغرات لإرسالها.")

return

result\_box.delete(1.0, tk.END)

progress\_bar["maximum"] = len(selected\_items)

progress\_bar["value"] = 0

def worker():

for idx, item\_id in enumerate(selected\_items, 1):

cve\_data = current\_cves[int(item\_id)]

reply = ask\_groq\_about\_cve(\*cve\_data)

result\_box.insert(tk.END, f"[+] CVE: {cve\_data[0]} | Host: {cve\_data[6]} | OS: {cve\_data[7]}\n{reply}\n{'-'\*60}\n")

progress\_bar["value"] = idx

root.update\_idletasks()

threading.Thread(target=worker).start()

def export\_results():

content = result\_box.get(1.0, tk.END)

if not content.strip():

messagebox.showinfo("تنبيه", "لا يوجد محتوى لحفظه.")

return

file\_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "\*.txt")])

if file\_path:

with open(file\_path, "w", encoding="utf-8") as f:

f.write(content)

messagebox.showinfo("تم الحفظ", "تم حفظ النتائج بنجاح.")

def send\_custom\_message():

msg = user\_input.get()

if not msg.strip():

return

user\_input.delete(0, tk.END)

os\_window = tk.Toplevel(root)

os\_window.title("اختيار نظام التشغيل")

os\_window.geometry("300x200")

tk.Label(os\_window, text="اختر نظام التشغيل المستهدف:").pack(pady=10)

os\_var = tk.StringVar(value="Windows")

for os\_option in ["Windows", "Linux", "macOS"]:

tk.Radiobutton(os\_window, text=os\_option, variable=os\_var, value=os\_option).pack(anchor=tk.W)

def confirm\_os():

os\_window.destroy()

threading.Thread(target=ask\_chat\_interactive, args=(msg, os\_var.get())).start()

tk.Button(os\_window, text="تأكيد", command=confirm\_os).pack(pady=10)

def ask\_chat\_interactive(user\_message, selected\_os):

headers = {

"Authorization": f"Bearer {groq\_api\_key}",

"Content-Type": "application/json"

}

prompt = (

f"You are a professional cybersecurity analyst.\n"

f"The user is working on a system running {selected\_os}.\n"

f"The following question or CVE-related inquiry was asked:\n\n"

f"{user\_message}\n\n"

f"Please provide accurate, detailed, and step-by-step instructions tailored ONLY for {selected\_os}.\n"

f"Assume the environment is production and upgrades may not be possible.\n"

f"Format your response using bullets, numbered steps, and code examples.\n"

f"Feel free to respond in the same language as the user input."

)

payload = {

"model": groq\_model,

"messages": [

{"role": "system", "content": "You are a cybersecurity expert."},

{"role": "user", "content": prompt}

],

"temperature": 0.7

}

try:

response = requests.post("https://api.groq.com/openai/v1/chat/completions", headers=headers, data=json.dumps(payload))

reply = response.json()["choices"][0]["message"]["content"] if response.status\_code == 200 else f"[Error {response.status\_code}]: {response.text}"

except Exception as e:

reply = f"[Exception]: {e}"

result\_box.insert(tk.END, f"👤 المستخدم:\n{user\_message}\n\n🤖 رد Groq:\n{reply}\n{'='\*60}\n")

root.update\_idletasks()

# GUI

root = tk.Tk()

root.title("Nessus CVE to Groq")

root.geometry("1100x800")

frame = tk.Frame(root)

frame.pack(pady=10)

tk.Button(frame, text="📁 افتح تقرير Nessus", command=open\_file).grid(row=0, column=0, padx=10)

tk.Button(frame, text="📤 أرسل المحدد لـ Groq", command=send\_selected\_to\_groq).grid(row=0, column=1, padx=10)

tk.Button(frame, text="💾 احفظ النتائج", command=export\_results).grid(row=0, column=2, padx=10)

columns = ("CVE", "Host", "OS", "Plugin", "Port", "Risk")

tree = ttk.Treeview(root, columns=columns, show="headings", selectmode="extended", height=12)

for col in columns:

tree.heading(col, text=col)

tree.column(col, width=150 if col != "CVE" else 180)

tree.pack(padx=10, pady=5, fill=tk.X)

search\_frame = tk.Frame(root)

search\_frame.pack(pady=10)

tk.Label(search\_frame, text="🔎 ابحث عن CVE:").pack(side=tk.LEFT)

search\_entry = tk.Entry(search\_frame, width=30)

search\_entry.pack(side=tk.LEFT, padx=5)

def search\_cve():

search\_term = search\_entry.get().strip().upper()

if not search\_term:

return

for item in tree.get\_children():

tree.delete(item)

for index, cve\_data in enumerate(current\_cves):

if search\_term in cve\_data[0].upper():

tree.insert("", tk.END, iid=index, values=(cve\_data[0], cve\_data[6], cve\_data[7], cve\_data[1], cve\_data[2], cve\_data[8]))

tk.Button(search\_frame, text="بحث", command=search\_cve).pack(side=tk.LEFT, padx=5)

progress\_bar = ttk.Progressbar(root, orient="horizontal", length=800, mode="determinate")

progress\_bar.pack(pady=5)

result\_box = scrolledtext.ScrolledText(root, wrap=tk.WORD, width=120, height=20)

result\_box.pack(padx=10, pady=10)

chat\_frame = tk.Frame(root)

chat\_frame.pack(pady=5)

user\_input = tk.Entry(chat\_frame, width=80)

user\_input.pack(side=tk.LEFT, padx=5)

tk.Button(chat\_frame, text="💬 استفسر من Groq", command=send\_custom\_message).pack(side=tk.LEFT)

root.mainloop()

**4.4.3 Database**

No database was used in this version; however, future versions may store CVE history or API usage logs.

**4.5 Issues and Problems Faced in the Project**

A screenshot of a black screen

AI-generated content may be incorrect.

**4.6 Testing**

**4.6.1 White Box Testing**

* Tested internal logic of functions like **extract\_cves\_from\_nessus**.
* Verified accurate parsing of sample **.nessus** files.

**4.6.2 Black Box Testing**

* End-to-end testing with various **.nessus** files.
* Validated that the correct number of CVEs was extracted and processed.

**4.6.3 Unit Testing**

* Mocked responses for **requests.post()** to test error handling.
* Wrote assertions for parsing edge cases.

**4.6.4 System Testing**

* Ran the complete workflow: open file → parse → send to Groq → display and export results.

**4.6.5 Acceptance Testing**

* Conducted with cybersecurity students who confirmed the tool’s usefulness and ease of use.

**4.6.6 Validation and Verification**

* Verified that the tool meets functional requirements.
* Validated that output is accurate and relevant to the input.

**4.7 Summary of the Chapter**

The tool was tested thoroughly through different testing methodologies. Minor bugs were resolved, and the application proved robust for typical use cases.

**Chapter 5: Challenges and Solutions**

**4.7.1 Block Listing**

No block listing feature is currently implemented, but future versions could include ignoring certain CVEs or hosts.

**4.7.3 Real-Time Detection Integration**

Currently, the tool works offline once the **.nessus** file is loaded. Real-time detection integration is not part of this version.

**4.7.5 GUI Responsiveness**

Fixed by offloading API calls to background threads using Python’s **threading**.

**4.7.6 Tools**

* Python 3.x
* Tkinter
* Groq API
* XML ElementTree
* Requests Library

**4.8.1 Programming and Libraries**

* Language: Python
* GUI: Tkinter
* Networking: Requests
* Data Parsing: XML.ElementTree
* Concurrency: Threading

**4.8.4 Development and Debugging**

Used standard Python debugging techniques along with print statements and unit tests to trace errors.

**4.8.6 Platform**

Developed and tested on Windows, but compatible with Linux and macOS.

**4.9 Results and Evaluation**

**4.9.1 Achievements and Findings**

* Successfully built a usable tool for automating CVE remediation research.
* Reduced analyst workload by providing actionable insights.

**4.9.2 Future Directions**

* Add multi-language support.
* Enable cloud-based scanning integration.
* Build a web version using Flask or Django.

**Chapter 6: Conclusion**

This project successfully bridges the gap between vulnerability scanning and remediation by leveraging AI to automate the interpretation of CVEs. The resulting application improves efficiency, reduces human error, and provides valuable insights directly from Nessus reports. While the current implementation is basic, it lays a strong foundation for future enhancements and broader adoption in cybersecurity workflows.

**References**

1. Groq API Documentation – <https://console.groq.com/docs>
2. Tkinter Official Documentation – <https://docs.python.org/3/library/tkinter.html>
3. Nessus Report Format – <https://www.tenable.com/products/nessus-vulnerability-scanner>
4. Python Requests Library – <https://docs.python-requests.org/en/latest/>
5. XML Processing in Python – <https://docs.python.org/3/library/xml.etree.elementtree.html>