# **AI Flask Web App - CyberAttack Analysis with AI and Block User**

A slightly-unique, production-ready Flask web app that demonstrates how AI can **monitor**, **detect**, and **auto‑mitigate** web attacks in real time. It combines lightweight heuristic detectors, rate limiting, and an **AI classifier** that explains why a request looks malicious and can generate a **ready-to-review mitigation script** (e.g., shell actions and NGINX snippets) for incident response. Styled with Tailwind (CDN) and designed for clarity and responsiveness.

## **Project Structure**

ai-guard-flask/  
├─ app.py  
├─ config.py  
├─ ai\_guard.py  
├─ detectors.py  
├─ rate\_limit.py  
├─ firewall.py  
├─ requirements.txt  
├─ .env.example  
├─ templates/  
│ ├─ base.html  
│ ├─ index.html  
│ ├─ dashboard.html  
│ └─ blocked.html  
└─ static/  
 └─ main.js

## **Quickstart**

1. **Clone & install**

* pip install -r requirements.txt  
  cp .env.example .env  
  *# then edit .env to set OPENAI\_API\_KEY, SECRET\_KEY (and optionally AI\_MODEL)*

1. **Run**

* flask --app app run --debug  
  *# or: python app.py*

1. **Visit**
   * App home: http://127.0.0.1:5000/
   * Security dashboard: http://127.0.0.1:5000/dashboard
2. **Simulate attacks** (for demo)
   * SQLi probe: http://127.0.0.1:5000/?q=' OR 1=1 --
   * XSS probe: http://127.0.0.1:5000/?q=<script>alert(1)</script>
   * LFI/path traversal: http://127.0.0.1:5000/?file=../../etc/passwd

## **Configuration**

**.env.example**

*# Flask*  
FLASK\_ENV=development  
SECRET\_KEY=change-me  
  
*# OpenAI*  
OPENAI\_API\_KEY=sk-your-key-here  
AI\_MODEL=gpt-4o-mini   
  
*# Guard settings*  
RATE\_LIMIT\_PER\_MIN=120  
RATE\_LIMIT\_BURST=30  
BLOCK\_DURATION\_SECONDS=900 *# 15 minutes*  
AI\_MIN\_CONFIDENCE=0.65 *# threshold to block based on AI verdict*

**requirements.txt**

Flask==3.0.3  
python-dotenv==1.0.1  
openai>=1.40.0  
requests>=2.32.3

## **Application Code**

### config.py

**import** os  
  
**class** Config:  
 SECRET\_KEY = os.getenv("SECRET\_KEY", "dev-change-me")  
  
 *# OpenAI*  
 OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY", "")  
 AI\_MODEL = os.getenv("AI\_MODEL", "gpt-4o-mini")  
 AI\_MIN\_CONFIDENCE = float(os.getenv("AI\_MIN\_CONFIDENCE", "0.65"))  
  
 *# Basic guard knobs*  
 RATE\_LIMIT\_PER\_MIN = int(os.getenv("RATE\_LIMIT\_PER\_MIN", "120"))  
 RATE\_LIMIT\_BURST = int(os.getenv("RATE\_LIMIT\_BURST", "30"))  
 BLOCK\_DURATION\_SECONDS = int(os.getenv("BLOCK\_DURATION\_SECONDS", "900"))  
  
 *# Misc*  
 ENV = os.getenv("FLASK\_ENV", "production")

### detectors.py

**import** re  
**from** urllib.parse **import** unquote  
  
*# Lightweight signature patterns (heuristic pre-filter)*  
SQLI = re.compile(r"**(**\bOR\b**|**\bAND\b**)**.\*?=**|**\bUNION\b**|**--**|**#**|**/\\***|**\bSELECT\b**|**\bDROP\b**|**\bINSERT\b**|**\bUPDATE\b", re.I)  
XSS = re.compile(r"<\s\*script**|**onerror\s\*=**|**onload\s\*=**|**javascript:\\S**|**<\s\*img**|**<\s\*svg**|**<\s\*iframe", re.I)  
LFI = re.compile(r"\.\./**|**\.\.\\**|**/etc/passwd**|**\\boot\.ini", re.I)  
RCE = re.compile(r"**(**;**|**\|\|**|**&&**)**\s\***(**cat**|**ls**|**id**|**uname**|**curl**|**wget**|**sh**|**bash**)**\b**|**`[^`]+`**|**\$\([^\)]+\)", re.I)  
SSRF = re.compile(r"https?://**(**127\.0\.0\.1**|**0\.0\.0\.0**|**localhost**|**169\.254\.169\.254**|**\[::1\]**)**", re.I)  
SCANNER\_UA = re.compile(r"sqlmap**|**nikto**|**nmap**|**curl**|**wget**|**acunetix**|**nessus", re.I)  
  
CATEGORIES = {  
 'SQLI': SQLI,  
 'XSS': XSS,  
 'LFI': LFI,  
 'RCE': RCE,  
 'SSRF': SSRF,  
}  
  
  
**def** extract\_surface(request):  
 *"""Extracts string surface from Flask request for heuristic and AI analysis."""*  
 parts = [  
 f"method={request.method}",  
 f"path={request.path}",  
 f"query={request.query\_string.decode('utf-8', 'ignore')}",  
 f"ip={request.remote\_addr}",  
 f"ua={request.headers.get('User-Agent','-')}",  
 ]  
 *# Safely include body (only small bodies)*  
 **try**:  
 **if** request.content\_length **and** request.content\_length < 4096:  
 body = request.get\_data(as\_text=True) **or** ""  
 parts.append(f"body={body}")  
 **except** Exception:  
 **pass**  
 **return** unquote("\n".join(parts))  
  
  
**def** heuristic\_assess(surface: str, user\_agent: str) -> dict:  
 score = 0.0  
 hits = []  
  
 **for** name, regex **in** CATEGORIES.items():  
 **if** regex.search(surface):  
 hits.append(name)  
 score += 0.35  
  
 **if** SCANNER\_UA.search(user\_agent **or** ""):  
 hits.append('SCANNER')  
 score += 0.3  
  
 *# Normalize score to 0..1*  
 score = min(1.0, score)  
 **return** {  
 'score': score,  
 'hits': hits,  
 }

### rate\_limit.py

**import** time  
**from** collections **import** defaultdict, deque  
  
**class** TokenBucketLimiter:  
 *"""Simple per-IP token bucket for rate limiting."""*  
 **def** \_\_init\_\_(self, per\_minute=120, burst=30):  
 self.capacity = burst  
 self.refill\_rate = per\_minute / 60.0 *# tokens per second*  
 self.buckets = defaultdict(**lambda**: {'tokens': burst, 'last': time.time()})  
  
 **def** allow(self, key: str) -> bool:  
 now = time.time()  
 b = self.buckets[key]  
 elapsed = now - b['last']  
 b['tokens'] = min(self.capacity, b['tokens'] + elapsed \* self.refill\_rate)  
 b['last'] = now  
 **if** b['tokens'] >= 1.0:  
 b['tokens'] -= 1.0  
 **return** True  
 **return** False

### ai\_guard.py

**import** json  
**from** typing **import** Dict, Any  
  
**from** openai **import** OpenAI  
  
**class** AIGuard:  
 **def** \_\_init\_\_(self, api\_key: str, model: str, min\_confidence: float = 0.65):  
 self.model = model  
 self.min\_confidence = min\_confidence  
 self.client = OpenAI(api\_key=api\_key) **if** api\_key **else** None  
  
 **def** classify(self, features: Dict[str, Any]) -> Dict[str, Any]:  
 *"""Use OpenAI to classify a request as BENIGN or MALICIOUS and explain why.*  
 *Returns: { verdict, confidence, categories, explanation }.*  
 *If API unavailable, returns a neutral verdict with low confidence.*  
 *"""*  
 **if** **not** self.client:  
 **return** {  
 'verdict': 'UNKNOWN',  
 'confidence': 0.0,  
 'categories': [],  
 'explanation': 'AI disabled (no API key).',  
 }  
  
 system = (  
 "You are a web security classifier. Classify HTTP requests as BENIGN or MALICIOUS. "  
 "Consider categories: SQLI, XSS, LFI, RCE, SSRF, SCANNER, AUTH\_BRUTE\_FORCE, OTHER. "  
 "Be conservative: only return MALICIOUS if you are reasonably confident."  
 )  
 user = (  
 "Analyze this request. Return compact JSON with keys: verdict (BENIGN|MALICIOUS), "  
 "confidence (0..1), categories (array of strings), explanation (<=30 words).\n\n"  
 f"RequestFeatures:\n{json.dumps(features, ensure\_ascii=False)}"  
 )  
  
 **try**:  
 resp = self.client.chat.completions.create(  
 model=self.model,  
 messages=[  
 {"role": "system", "content": system},  
 {"role": "user", "content": user},  
 ],  
 temperature=0.1,  
 response\_format={"type": "json\_object"}  
 )  
 content = resp.choices[0].message.content  
 data = json.loads(content)  
 **return** {  
 'verdict': data.get('verdict', 'UNKNOWN'),  
 'confidence': float(data.get('confidence', 0.0)),  
 'categories': data.get('categories', []) **or** [],  
 'explanation': data.get('explanation', ''),  
 }  
 **except** Exception **as** e:  
 **return** {  
 'verdict': 'UNKNOWN',  
 'confidence': 0.0,  
 'categories': [],  
 'explanation': f'AI error: {e}',  
 }  
  
 **def** should\_block(self, ai\_result: Dict[str, Any]) -> bool:  
 **return** ai\_result.get('verdict') == 'MALICIOUS' **and** ai\_result.get('confidence', 0.0) >= self.min\_confidence

### firewall.py

**import** json  
**from** datetime **import** datetime  
**from** openai **import** OpenAI  
  
TEMPLATE\_BASELINE = """#!/usr/bin/env bash  
# Generated at {ts}  
# Purpose: Apply temporary mitigation for suspected attack  
IP="{ip}"  
DURATION\_MIN=15  
  
echo "Blocking $IP for $DURATION\_MIN minutes via iptables (example; review before applying)"  
# Example iptables commands (review! may vary by distro)  
iptables -I INPUT -s "$IP" -j DROP  
# Optional: schedule unblock  
# (crontab) echo "(sleep $((DURATION\_MIN\*60)); iptables -D INPUT -s $IP -j DROP) &" | at now  
  
# Example NGINX snippet to drop obvious malicious payloads  
cat > /etc/nginx/conf.d/ai-guard-snippet.conf <<'NGINX'  
map $request\_uri $ai\_guard\_block {  
 default 0;  
 ~\*(<script>|select**\**s+.\*from|**\**.{2}/|/etc/passwd) 1;  
}  
server {  
 if ($ai\_guard\_block) { return 403; }  
}  
NGINX  
  
systemctl reload nginx 2>/dev/null || true  
"""  
  
**class** MitigationGenerator:  
 **def** \_\_init\_\_(self, api\_key: str, model: str):  
 self.client = OpenAI(api\_key=api\_key) **if** api\_key **else** None  
 self.model = model  
  
 **def** generate(self, incident: dict) -> str:  
 *"""Ask AI to produce a short shell script + config hints tailored to the incident.*  
 *Falls back to a baseline template if AI is unavailable.*  
 *"""*  
 ip = incident.get('ip', '0.0.0.0')  
 ts = datetime.utcnow().isoformat()  
 **if** **not** self.client:  
 **return** TEMPLATE\_BASELINE.format(ip=ip, ts=ts)  
  
 prompt = (  
 "You are a defensive security assistant. Generate a short bash script (with comments) "  
 "that a human can review to apply a \*\*temporary\*\* mitigation for the web attack described. "  
 "NEVER include offensive commands. Prefer safe blocks, IP drop, and NGINX/WAF snippets. "  
 "Keep it under ~80 lines. Include inline comments and reminders to review and roll back.\n\n"  
 f"Incident (JSON):\n{json.dumps(incident, ensure\_ascii=False, indent=2)}"  
 )  
  
 **try**:  
 resp = self.client.chat.completions.create(  
 model=self.model,  
 messages=[  
 {"role": "system", "content": "You help defenders mitigate web attacks safely."},  
 {"role": "user", "content": prompt},  
 ],  
 temperature=0.2,  
 )  
 **return** resp.choices[0].message.content.strip()  
 **except** Exception:  
 **return** TEMPLATE\_BASELINE.format(ip=ip, ts=ts)

### app.py

**import** os  
**import** time  
**import** sqlite3  
**from** functools **import** wraps  
**from** datetime **import** datetime, timedelta  
  
**from** flask **import** (  
 Flask, request, render\_template, redirect, url\_for, abort, g, jsonify, flash  
)  
**from** dotenv **import** load\_dotenv  
  
**from** config **import** Config  
**from** detectors **import** extract\_surface, heuristic\_assess  
**from** rate\_limit **import** TokenBucketLimiter  
**from** ai\_guard **import** AIGuard  
**from** firewall **import** MitigationGenerator  
  
load\_dotenv()  
config = Config()  
app = Flask(\_\_name\_\_)  
app.config.from\_object(config)  
  
limiter = TokenBucketLimiter(  
 per\_minute=config.RATE\_LIMIT\_PER\_MIN,  
 burst=config.RATE\_LIMIT\_BURST,  
)  
ai\_guard = AIGuard(  
 api\_key=config.OPENAI\_API\_KEY,  
 model=config.AI\_MODEL,  
 min\_confidence=config.AI\_MIN\_CONFIDENCE,  
)  
mitigator = MitigationGenerator(  
 api\_key=config.OPENAI\_API\_KEY,  
 model=config.AI\_MODEL,  
)  
  
BLOCKLIST = {} *# ip -> unblock\_time (epoch seconds)*  
DB\_PATH = os.path.join(os.path.dirname(\_\_file\_\_), 'incidents.db')  
  
*# DB*  
**def** get\_db():  
 **if** 'db' **not** **in** g:  
 g.db = sqlite3.connect(DB\_PATH)  
 g.db.row\_factory = sqlite3.Row  
 **return** g.db  
  
@app.teardown\_appcontext  
**def** close\_db(error=None):  
 db = g.pop('db', None)  
 **if** db **is** **not** None:  
 db.close()  
  
**with** app.app\_context():  
 db = get\_db()  
 db.execute(  
 """  
 CREATE TABLE IF NOT EXISTS incidents (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 ts TEXT NOT NULL,  
 ip TEXT,  
 path TEXT,  
 method TEXT,  
 user\_agent TEXT,  
 surface TEXT,  
 heuristic\_score REAL,  
 heuristic\_hits TEXT,  
 ai\_verdict TEXT,  
 ai\_confidence REAL,  
 ai\_categories TEXT,  
 ai\_explanation TEXT,  
 action TEXT,  
 mitigation\_script TEXT  
 );  
 """  
 )  
 db.commit()  
  
  
**def** is\_blocked(ip: str) -> bool:  
 now = time.time()  
 until = BLOCKLIST.get(ip)  
 **if** until **and** now < until:  
 **return** True  
 **elif** until **and** now >= until:  
 BLOCKLIST.pop(ip, None)  
 **return** False  
  
  
**def** block\_ip(ip: str):  
 BLOCKLIST[ip] = time.time() + app.config['BLOCK\_DURATION\_SECONDS']  
  
  
**def** log\_incident(\*\*kwargs):  
 db = get\_db()  
 db.execute(  
 """INSERT INTO incidents (  
 ts, ip, path, method, user\_agent, surface,  
 heuristic\_score, heuristic\_hits, ai\_verdict, ai\_confidence,  
 ai\_categories, ai\_explanation, action, mitigation\_script  
 ) VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?)""",  
 (  
 datetime.utcnow().isoformat(),  
 kwargs.get('ip'),  
 kwargs.get('path'),  
 kwargs.get('method'),  
 kwargs.get('user\_agent'),  
 kwargs.get('surface'),  
 kwargs.get('heuristic\_score', 0.0),  
 ",".join(kwargs.get('heuristic\_hits', [])),  
 kwargs.get('ai\_verdict'),  
 kwargs.get('ai\_confidence', 0.0),  
 ",".join(kwargs.get('ai\_categories', [])),  
 kwargs.get('ai\_explanation'),  
 kwargs.get('action'),  
 kwargs.get('mitigation\_script'),  
 )  
 )  
 db.commit()  
  
  
@app.before\_request  
**def** guard\_request():  
 *# Skip for health/static/dashboard endpoints to avoid locking yourself out*  
 safe\_paths = {'/dashboard', '/static', '/blocked'}  
 **if** any(request.path.startswith(p) **for** p **in** safe\_paths):  
 **return**  
  
 ip = request.headers.get('X-Forwarded-For', request.remote\_addr)  
 **if** is\_blocked(ip):  
 **return** redirect(url\_for('blocked'))  
  
 *# Rate limiting*  
 **if** **not** limiter.allow(ip):  
 block\_ip(ip)  
 log\_incident(  
 ip=ip,  
 path=request.path,  
 method=request.method,  
 user\_agent=request.headers.get('User-Agent','-'),  
 surface='(rate limit exceeded)',  
 heuristic\_score=1.0,  
 heuristic\_hits=['RATE\_LIMIT'],  
 ai\_verdict='MALICIOUS',  
 ai\_confidence=1.0,  
 ai\_categories=['RATE\_LIMIT'],  
 ai\_explanation='Excessive requests from single IP.',  
 action='blocked',  
 mitigation\_script='(auto) temporary IP block',  
 )  
 **return** redirect(url\_for('blocked'))  
  
 *# Heuristic prefilter*  
 surface = extract\_surface(request)  
 heur = heuristic\_assess(surface, request.headers.get('User-Agent','-'))  
  
 *# If clearly suspicious or random sample, ask AI to classify*  
 ai\_result = {'verdict': 'UNKNOWN', 'confidence': 0.0, 'categories': [], 'explanation': ''}  
 **if** heur['score'] >= 0.35 **or** request.args.get('force\_ai') == '1':  
 features = {  
 'ip': ip,  
 'method': request.method,  
 'path': request.path,  
 'query': request.args.to\_dict(flat=False),  
 'headers': {k: v **for** k, v **in** request.headers.items() **if** k.lower() **in** ['user-agent', 'referer', 'content-type']},  
 'body\_len': request.content\_length **or** 0,  
 'surface': surface[:1500],  
 }  
 ai\_result = ai\_guard.classify(features)  
  
 *# Decide action*  
 action = 'allow'  
 **if** heur['score'] >= 0.8 **or** ai\_guard.should\_block(ai\_result):  
 action = 'block'  
  
 **if** action == 'block':  
 block\_ip(ip)  
 *# Ask AI to produce a small mitigation script for incident response*  
 incident = {  
 'ip': ip,  
 'path': request.path,  
 'method': request.method,  
 'user\_agent': request.headers.get('User-Agent','-'),  
 'heuristic\_hits': heur['hits'],  
 'ai': ai\_result,  
 }  
 script = mitigator.generate(incident)  
 log\_incident(  
 ip=ip,  
 path=request.path,  
 method=request.method,  
 user\_agent=request.headers.get('User-Agent','-'),  
 surface=surface,  
 heuristic\_score=heur['score'],  
 heuristic\_hits=heur['hits'],  
 ai\_verdict=ai\_result.get('verdict'),  
 ai\_confidence=ai\_result.get('confidence'),  
 ai\_categories=ai\_result.get('categories'),  
 ai\_explanation=ai\_result.get('explanation'),  
 action='blocked',  
 mitigation\_script=script,  
 )  
 **return** redirect(url\_for('blocked'))  
  
 *# Otherwise allow and optionally log suspicious-but-allowed*  
 **if** heur['score'] >= 0.35:  
 log\_incident(  
 ip=ip,  
 path=request.path,  
 method=request.method,  
 user\_agent=request.headers.get('User-Agent','-'),  
 surface=surface,  
 heuristic\_score=heur['score'],  
 heuristic\_hits=heur['hits'],  
 ai\_verdict=ai\_result.get('verdict'),  
 ai\_confidence=ai\_result.get('confidence'),  
 ai\_categories=ai\_result.get('categories'),  
 ai\_explanation=ai\_result.get('explanation'),  
 action='allowed',  
 mitigation\_script='',  
 )  
  
*# Routes*   
@app.route('/')  
**def** index():  
 **return** render\_template('index.html')  
  
@app.route('/blocked')  
**def** blocked():  
 **return** render\_template('blocked.html')  
  
@app.route('/dashboard')  
**def** dashboard():  
 db = get\_db()  
 rows = db.execute("SELECT \* FROM incidents ORDER BY id DESC LIMIT 200").fetchall()  
 **return** render\_template('dashboard.html', rows=rows, blocklist=BLOCKLIST)  
  
@app.route('/dashboard/unblock/<ip>', methods=['POST'])  
**def** unblock(ip):  
 BLOCKLIST.pop(ip, None)  
 flash(f"Unblocked {ip}")  
 **return** redirect(url\_for('dashboard'))  
  
@app.route('/echo')  
**def** echo():  
 *# innocuous endpoint to test rate limiting / AI analysis*  
 msg = request.args.get('msg', 'hello')  
 **return** jsonify({'ok': True, 'echo': msg})  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=os.getenv('FLASK\_ENV') == 'development')

## **Templates (Tailwind + simple UI)**

### templates/base.html

<!doctype html>  
<**html** lang="en">  
 <**head**>  
 <**meta** charset="utf-8" />  
 <**meta** name="viewport" content="width=device-width, initial-scale=1" />  
 <**title**>AI Guard</**title**>  
 <**script** src="https://cdn.tailwindcss.com"></**script**>  
 </**head**>  
 <**body** class="bg-slate-50 text-slate-900">  
 <**header** class="bg-white border-b sticky top-0 z-10">  
 <**div** class="max-w-6xl mx-auto px-4 py-3 flex items-center justify-between">  
 <**a** href="/" class="font-semibold text-xl">🛡️ AI Guard</**a**>  
 <**nav** class="space-x-4 text-sm">  
 <**a** href="/" class="hover:underline">Home</**a**>  
 <**a** href="/dashboard" class="hover:underline">Security Dashboard</**a**>  
 </**nav**>  
 <**div** class="flex items-center gap-2">  
 <**span** class="h-2 w-2 rounded-full bg-emerald-500 inline-block"></**span**>  
 <**span** class="text-xs text-slate-500">AI Shield ON</**span**>  
 </**div**>  
 </**div**>  
 </**header**>  
 <**main** class="max-w-6xl mx-auto px-4 py-8">  
 {% block content %}{% endblock %}  
 </**main**>  
 <**script** src="/static/main.js"></**script**>  
 </**body**>  
</**html**>

### templates/index.html

{% extends 'base.html' %}  
{% block content %}  
<**div** class="grid gap-6 md:grid-cols-2">  
 <**section** class="bg-white rounded-2xl shadow p-6">  
 <**h1** class="text-2xl font-semibold mb-2">Welcome</**h1**>  
 <**p** class="text-slate-600">This app demonstrates an AI-augmented Web Application Firewall (WAF): heuristic pre-filtering, rate limiting, and an AI classifier that explains and suggests safe mitigations.</**p**>  
 <**ul** class="list-disc ml-6 mt-4 text-slate-600">  
 <**li**>Try hitting the home URL with suspicious query strings (see Quickstart).</**li**>  
 <**li**>Open the <**a** class="underline" href="/dashboard">Security Dashboard</**a**> to see detections.</**li**>  
 </**ul**>  
 </**section**>  
 <**section** class="bg-white rounded-2xl shadow p-6">  
 <**h2** class="text-xl font-semibold mb-3">Quick Probe</**h2**>  
 <**form** id="probeForm" class="flex gap-2">  
 <**input** name="msg" id="msg" class="w-full border rounded-xl px-3 py-2" placeholder="type something benign or suspicious..." />  
 <**button** class="px-4 py-2 rounded-xl bg-slate-900 text-white">Send</**button**>  
 </**form**>  
 <**pre** id="probeOut" class="mt-3 text-sm bg-slate-50 border rounded-xl p-3 overflow-x-auto"></**pre**>  
 </**section**>  
</**div**>  
{% endblock %}

### templates/dashboard.html

{% extends 'base.html' %}  
{% block content %}  
<**h1** class="text-2xl font-semibold mb-4">Security Dashboard</**h1**>  
<**div** class="grid gap-6 md:grid-cols-3">  
 <**div** class="bg-white rounded-2xl shadow p-5">  
 <**h2** class="font-semibold mb-2">Blocklist</**h2**>  
 <**ul** class="text-sm space-y-1">  
 {% for ip, until in blocklist.items() %}  
 <**li** class="flex items-center justify-between">  
 <**span** class="font-mono">{{ ip }}</**span**>  
 <**form** method="post" action="/dashboard/unblock/{{ ip }}">  
 <**button** class="text-xs px-2 py-1 rounded bg-emerald-600 text-white">Unblock</**button**>  
 </**form**>  
 </**li**>  
 {% else %}  
 <**li** class="text-slate-500">No IPs blocked.</**li**>  
 {% endfor %}  
 </**ul**>  
 </**div**>  
 <**div** class="md:col-span-2 bg-white rounded-2xl shadow p-5 overflow-x-auto">  
 <**h2** class="font-semibold mb-3">Recent Incidents</**h2**>  
 <**table** class="min-w-full text-sm">  
 <**thead** class="text-left text-slate-500">  
 <**tr**>  
 <**th** class="py-2 pr-4">When</**th**>  
 <**th** class="py-2 pr-4">IP</**th**>  
 <**th** class="py-2 pr-4">Path</**th**>  
 <**th** class="py-2 pr-4">Heur.</**th**>  
 <**th** class="py-2 pr-4">AI</**th**>  
 <**th** class="py-2 pr-4">Action</**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 {% for r in rows %}  
 <**tr** class="border-t">  
 <**td** class="py-2 pr-4 whitespace-nowrap">{{ r['ts'] }}</**td**>  
 <**td** class="py-2 pr-4 font-mono">{{ r['ip'] }}</**td**>  
 <**td** class="py-2 pr-4">{{ r['method'] }} {{ r['path'] }}</**td**>  
 <**td** class="py-2 pr-4">{{ '%.2f'|format(r['heuristic\_score']) }} ({{ r['heuristic\_hits'] }})</**td**>  
 <**td** class="py-2 pr-4">{{ r['ai\_verdict'] }} {{ '%.2f'|format(r['ai\_confidence'] or 0) }}<**br**><**span** class="text-slate-500">{{ r['ai\_categories'] }}</**span**></**td**>  
 <**td** class="py-2 pr-4">{{ r['action'] }}</**td**>  
 </**tr**>  
 <**tr** class="bg-slate-50">  
 <**td** colspan="6" class="p-3">  
 <**details**>  
 <**summary** class="cursor-pointer text-slate-700">Details **&** Mitigation Script</**summary**>  
 <**div** class="grid md:grid-cols-2 gap-3 mt-2">  
 <**pre** class="bg-white border rounded-xl p-3 overflow-x-auto"><**strong**>Surface</**strong**>\n{{ r['surface'] }}</**pre**>  
 <**pre** class="bg-white border rounded-xl p-3 overflow-x-auto"><**strong**>Mitigation</**strong**>\n{{ r['mitigation\_script'] }}</**pre**>  
 </**div**>  
 </**details**>  
 </**td**>  
 </**tr**>  
 {% else %}  
 <**tr**><**td** colspan="6" class="py-6 text-center text-slate-500">No incidents yet.</**td**></**tr**>  
 {% endfor %}  
 </**tbody**>  
 </**table**>  
 </**div**>  
</**div**>  
{% endblock %}

### templates/blocked.html

{% extends 'base.html' %}  
{% block content %}  
<**div** class="max-w-lg mx-auto bg-white rounded-2xl shadow p-6 text-center">  
 <**div** class="text-5xl mb-3">🚫</**div**>  
 <**h1** class="text-xl font-semibold">Request Blocked</**h1**>  
 <**p** class="text-slate-600 mt-2">Your IP has been temporarily blocked due to suspicious activity. If you believe this is a mistake, please contact the site owner.</**p**>  
 <**p** class="text-xs text-slate-500 mt-4">The block will expire automatically after a short period.</**p**>  
</**div**>  
{% endblock %}

### static/main.js

**const** f = document.querySelector('#probeForm');  
**if** (f) {  
 f.addEventListener('submit', **async** (e) **=>** {  
 e.preventDefault();  
 **const** msg = document.querySelector('#msg').value || '';  
 **const** res = **await** fetch(`/echo?msg=${encodeURIComponent(msg)}`);  
 **const** data = **await** res.json();  
 document.querySelector('#probeOut').textContent = JSON.stringify(data, **null**, 2);  
 });  
}

## **How It Works (Architecture)**

1. **before\_request middleware** extracts a compact *surface* of the HTTP request.
2. **Heuristic detectors** (regex signatures) produce a quick score and category hits (SQLI, XSS, etc.).
3. If the score is above a threshold, the request is sent to **OpenAI** for classification; the model returns a JSON verdict with confidence and categories.
4. If either the heuristics are very high or the **AI verdict is MALICIOUS** with sufficient confidence, the IP is **temporarily blocked** and an **incident** is written to SQLite (with an **AI‑generated mitigation script** attached for the analyst to review).
5. The **Security Dashboard** shows the last incidents, heuristic/AI details, and mitigation snippets. You can unblock IPs.

## **Deployment (production sketch)**

* Run behind a reverse proxy (NGINX/Traefik) and set X-Forwarded-For correctly.
* Use **gunicorn**: gunicorn -w 4 -b 0.0.0.0:8000 app:app.
* Persist the SQLite file and logs to durable storage; prefer a managed DB for scale.
* Restrict dashboard access (e.g., VPN/basic auth) and set a strong SECRET\_KEY.
* Consider exporting incidents to SIEM (e.g., via webhook).

## **Testing Ideas**

* SQLi: ?q=' OR 1=1 --
* XSS: ?q=<svg onload=alert(1)>
* LFI: ?file=../../etc/passwd
* SSRF-like URL: ?u=http://127.0.0.1:80/
* Scanner UA: curl -A sqlmap http://localhost:5000/

## **Notes**

* **Safety**: Mitigation scripts are examples only and are not executed automatically. Always review and apply in a safe, controlled manner.
* **Model**: Default is gpt-4o-mini for cost/speed. You can set AI\_MODEL to other supported chat models.
* **Extensibility**: Add detectors in detectors.py, tune thresholds, and forward incidents to Slack/SIEM.

## **How to test on local server:**

## **Trigger detections (in your browser or curl)**

**SQLi probe**

http://127.0.0.1:5000/?q=' OR 1=1 --

**XSS probe**

http://127.0.0.1:5000/?q=<script>alert(1)</script>

**LFI probe**

http://127.0.0.1:5000/?file=../../etc/passwd

Now open **/dashboard** and check the incident table for heuristic hits and (if AI key set) an AI verdict.

## **Force AI on a benign request**

If you want to see the AI classifier run even for a normal request:

http://127.0.0.1:5000/?force\_ai=1

Dashboard ➜ you’ll see ai\_verdict, ai\_confidence, and a short explanation.

## **Guaranteed block via heuristics (no AI key needed)**

Hit the app with **multiple attack signatures** in one request (SQLi + XSS + LFI ≥ 0.8 score):

http://127.0.0.1:5000/?q=' OR 1=1 --&payload=<script>alert(1)</script>&file=../../etc/passwd

You should be redirected to **/blocked** immediately, and an incident will appear on **/dashboard** with action=blocked.