



by Sergey Egorov

VA / Pentest Use Cases for LLM

1. **Static** analysis – Check the source code, configuration / infra as code, dependencies.
2. **Dynamic** analysis – Analyze HTTP requests/responses of running web apps, identify dependency/middleware versions, admin panels, parse and generate test requests for APIs.
3. **Infrastructure** scanning – Search CVEs, exploits for services' versions.
4. **Scanner rules** and templates generation.
5. **Reducing false positives** – Cross checks, evaluation of code flow.
6. Vulnerabilities **description** generation.
7. VM / **Pentest report** / generation/ enrichment.

Classic VA Challenges

- Evaluating bad patterns and writing rules / templates for specific languages / CVEs -> **Slow, Low Coverage**
- Most rules / templates are signature based (aka regex) -> **False Positives**
- No rules for your language/ framework / version / CVE -> **False Negatives**
- Code Flow / Semantic analysis requires deep knowledge of languages, frameworks, vulnerability patterns -> **Expensive**
- Writing vulnerabilities and pentest reports -> **Tedious**

How LLM can help with this?

- **Analyses code** and other user supplied data.
- **Knows bad patterns**, identifies popular technologies, versions, known vulnerabilities, configuration specifics.
- Aware of **CVEs**.
- Writes **vulnerabilities descriptions** and other report parts (management summary).
- Provides the output in specified format (i.e. **json/xml/markdown**) to aggregate, parse and process.

* assuming the model training date, size and quality

LLM Usage Options

✓ Cloud – OpenAI, Claude, Qwen etc.

Perfect for self-use or non-sensitive data

✓ Local

✓ “Private” cloud instances

Azure OpenAI, GCP Gemini etc.



Run
locally

Use
SaaS

Challenges with LLM usage

- **Price** - pay-as-you-go or bring-your-own-GPU.
- **Speed** - limitations for fuzzing and tasks requiring quick replies.
- **Limited context** – normally up to 131k tokens per “session”, max up to 10 million (Gemini 1.5 Pro - 2 mil, Llama 4 - 10 mil). Bigger context -> more GPU required.
- **Hallucinations** - generates fiction, irrelevant content, which looks trustful.
- **Consistency** - different output for the same query, broken output formats.
- **Security** - tends to process any user input, leading to prompt injections and hijacking input flows.

Deploying Local LLM

- <https://ollama.com/> - servers
- <https://lmstudio.ai/> - workstations
- 24 GB+ VRAM for ~**30B models** for consistent results.
 - max ~35k tokens context fit in 24GB for this model size.
 - ~10GB VRAM for each extra ~30k tokens.
- “Instruct” models are faster.
- “Reasoning” – more accurate.

OpenAI API



```
curl http://localhost:11434/v1/chat/completions \
-H 'Content-Type: application/json' \
-d '{
  "model": "llama2",
  "messages": [
    {
      "role": "user",
      "content": "Hello!"
    }
  ]
}'
```

<https://ollama.com/blog/openai-compatibility>
<https://lmstudio.ai/docs/app/api/endpoints/openai>

OpenAI Python library

```
from openai import OpenAI

client = OpenAI(
    base_url = 'http://localhost:11434/v1',
    api_key='ollama', # required, but unused
)

response = client.chat.completions.create(
    model="llama2",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The LA Dodgers won in 2020."},
        {"role": "user", "content": "Where was it played?"}
    ]
)

print(response.choices[0].message.content)
```


Code Analysis with LLM

Scenario 1

1. Examine the repo, **pick files** worth checking (by file extension, content patterns).
2. **Ship each file to LLM**, retrieve the reply in specific format (json, xml, etc.)
3. **Verify** the issues against the same or other LLM.
4. **Aggregate** the results.

Scenario 2

1. Prepare a **single file archive**, including complete repository structure.
2. Ship it to LLM, ensure the **code flow** is also evaluated to reduce false positives.
3. **Verify** the issues.
4. **Parse** the result.

SAST Results Comparison for Java, Python and C# applications

		ScanSuite SAST	Qwen 2.5 Coder Instruct 32B	Commercial SAST X	Commercial IDE Plugin X
Application Name	Vulnado	https://github.com/ScaleSec/vulnado			
Programming language, framework	Java				
Known High Risk vulnerabilities in the application	5 (SQLi, RCE, XSS, CSRF, SSRF)				
Detected High Risk True Positives / % of total		5 / 100%	5 / 100%	2 / 40%	2 / 40%
Detected True Positive vulnerability types (any risk level)		SQLi, RCE, XSS, CSRF, SSRF, Weak Hash	SQLi, RCE, XSS, CSRF, SSRF, Weak Hash	SQLi, RCE	SQLi, RCE
Undetected vulnerability types (any risk level)		-	-	XSS, CSRF, SSRF, Weak Hash	XSS, CSRF, SSRF, Weak Hash
Application Name	django.nv	https://github.com/NetSPI/django.nv			
Programming language, framework	Python / Django				
Known High Risk vulnerabilities in the application	8 (SQLi, RCE, IDOR, CSRF, Authorization Bypass, Broken Auth and Session Mgmt, Debug console, Hardcoded secret for JWT)				
Detected High Risk True Positives / % of total		3 / 37%	6 / 75%	3 / 37%	3 / 37%
Detected True Positive vulnerability types (any risk level)		SQLi, RCE, Hardcoded secret for JWT, CSRF, XSS, Open Redirect	RCE, IDOR, Broken Auth and Session Mgmt, Hardcoded secret for JWT, CSRF, Debug console, Insecure Session Cookie, Insecure Session Storage and Serialization	SQLi, RCE, Hardcoded secret for JWT, HTTP Response Splitting, XSS, Open Redirect	SQLi, RCE, Hardcoded secret, HTTP Resp Split, XSS, Open Redirect
Undetected vulnerability types (any risk level)		IDOR, Authorisation bypass, Broken Auth and Session Mgmt, HTTP Resp Split, Debug console	SQLi, XSS - both found by GPT-4 HTTP Response Splitting	IDOR, Authorisation bypass, Broken Auth and Session Mgmt, CSRF, Debug console	IDOR, Authorisation bypass, Broken Auth and Session Mgmt, CSRF, Debug console
Application Name	dvcsharp-api	https://github.com/appsecco/dvcsharp-api			
Programming language, framework	C#				
Known High Risk vulnerabilities in the application	7 (SQLi, XXE, SSRF, Auth Bypass via Insecure JWT and SSO Cookie, Authorization Bypass, Insecure Deserialization, Weak password reset)				
Detected High Risk True Positives / % of total		5 / 71%	6 / 86%	3 / 43%	lang not supported
Detected True Positive vulnerability types (any risk level)		SQLi, XXE, CSRF, Authentication Bypass via Insecure JWT and SSO Cookie, Insecure Deserialization, Missing HSTS, Weak password reset	SQLi, XXE, SSRF, CSRF, Authentication Bypass via Insecure JWT and SSO Cookie, Insecure Deserialization, Weak password reset	SQLi, XXE, SSRF, Insecure Deserialization	-
Undetected vulnerability types (any risk level)		Authorization Bypass, SSRF	Authorization Bypass	Authentication Bypass via Insecure JWT and SSO Cookie, Authorization Bypass, Weak password reset, Missing HSTS	

Code Analysis Demo

Scan History

Products

Static Code Analysis

Web App Scanning

Infrastructure Checks

Upload Report

My Rules

Credentials

Vuln DB

Logout

Scanning mode

Once

Branch name

main

Main language

python

Product / Eng ID

Test Static (2)

SAST

Semgrep

GitLab

CodeQL

Semgrep (My Rules)

External:

Semgrep Pro

Snyk

ML SAST

Secrets Scan

GitLeaks

Trufflehog

Parker

Dependency Checks

Trivy

OSV Scanner

Infra as Code Scan

KICS

Submit

<https://scansuite.gitbook.io/scansuite/usage/performing-a-static-code-analysis/executing-the-openai-scan>

Vulnerabilities Generation Examples. SAST.

Vulnerability Name	Description	File Name	Vulnerable Parameter	Code Snippet	Fixed Code
Open Redirect	The method links and linksV2 are vulnerable to open redirect attacks as they directly use the user-supplied URL parameter without validation.	./src/main/java/com/scalesec/vulnado/LinksController.java	url	<pre>@RequestMapping(value = "/links", produces = "application/json") List<String> links(@RequestParam String url) throws IOException{ return LinkLister.getLinks(url); } @RequestMapping(value = "/links-v2", produces = "application/json") List<String> linksV2(@RequestParam String url) throws BadRequest{ return LinkLister.getLinksV2(url); }</pre>	<pre>@RequestMapping(value = "/links", produces = "application/json") List<String> links(@RequestParam String url) throws IOException{ if (!isValidURL(url)) throw new BadRequestException("Invalid URL"); return LinkLister.getLinks(url); } @RequestMapping(value = "/links-v2", produces = "application/json") List<String> linksV2(@RequestParam String url) throws BadRequest{ if (!isValidURL(url)) throw new BadRequestException("Invalid URL"); return LinkLister.getLinksV2(url); }</pre>
Improper Exception Handling	The method linksV2 throws a custom exception, which is not defined in the provided code. It seems like a typo and should be replaced with an actual exception class or handled appropriately.	./src/main/java/com/scalesec/vulnado/LinksController.java	@RequestParam String url	<pre>List<String> linksV2(@RequestParam String url) throws BadRequest{ return LinkLister.getLinksV2(url); }</pre>	<pre>List<String> linksV2(@RequestParam String url) throws IOException{ if (!isValidURL(url)) throw new IllegalArgumentException("Invalid URL"); return LinkLister.getLinksV2(url); }</pre>
SQL Injection	The SQL query is constructed using string concatenation, which allows for injection attacks.	./src/main/java/com/scalesec/vulnado/User.java	un (username)	<pre>String query = "select * from users where username = '" + un + "' limit 1";</pre>	<pre>String query = "SELECT * FROM users WHERE username = ? LIMIT 1"; PreparedStatement pstmt.setString(1, un); ResultSet rs = pstmt.executeQuery();</pre>

Analyzing Web / API traffic

Scenario 1

1. Crawl the application, form the sitemap, record requests / responses. Save JS files.
2. Send the request/response pairs and JS files to LLM.
3. Identify dependency / middleware versions, admin panels.
4. Generate test requests for suspicious parameters.
5. Parse and aggregate the replies.

Scenario 2

1. Send specific request-response pairs straight from Proxy / Scanner to LLM for manual examination.

Scenario 3 (API analysis)

1. Parse given OpenAPI / GraphQL, generate testing data.
2. Run the tests, send request-response pairs to LLM for analysis.

Web Scanning Demo

The screenshot displays the Scansuite web application interface. On the left is a dark sidebar with navigation links: Scan History, Products, Static Code Analysis, Web App Scanning (highlighted with a red box), Infrastructure Checks, Upload Report, My Rules, Credentials, Vuln DB, and Logout. The main content area is white and contains configuration options for a web scan. At the top, there are dropdowns for 'Run now' and a time limit of '00:30'. Below these are fields for 'Product / Eng ID' (set to 'Test 2 (3)'), 'Cookies (optional)' (set to 'cookie1=asd, cookie2=qwe'), and 'Custom header (optional)' (set to 'Authorization: Basic ...'). The bottom section features three columns of toggle switches for different scanning methods: 'Quick' (ZAP base, SSLyze, Dastardly), 'My Rules' (Nuclei custom), 'Balanced' (Arachni, Wapiti, Nuclei), 'Pro / Licensed' (Nessus), 'Deep / Specialized' (Gobuster, ZAP full, WordPress, ML DAST - highlighted with a red box), and 'API Scan' (ZAP OpenAPI, Wapiti OpenAPI). A 'Submit' button is located at the bottom right.

Run now ▼ 00:30 ▼

Product / Eng ID Test 2 (3) ▼ Cookies (optional) cookie1=asd, cookie2=qwe

Custom header (optional) Authorization: Basic ...

Quick

- ☐ ZAP base
- ☐ SSLyze
- ☐ Dastardly

My Rules

- ☐ Nuclei custom

Balanced

- ☐ Arachni
- ☐ Wapiti
- ☐ Nuclei

Pro / Licensed

- ☐ Nessus

Deep / Specialized

- ☐ Gobuster
- ☐ ZAP full
- ☐ WordPress
- ☒ ML DAST

API Scan

- ☐ ZAP OpenAPI
- ☐ Wapiti OpenAPI

Submit

<https://scansuite.gitbook.io/scansuite/usage/performing-dynamic-web-scanning>

ML DAST Sample Vulnerabilities.

Information Disclosure	The server reveals unnecessary information about its configuration through headers like 'X-Powered-By'.	/	X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1	Disable or remove the X-Powered-By header and other unnecessary information disclosure in server responses.
Potential SQL Injection	The 'cat' parameter in the URL might be vulnerable to SQL injection attacks.	/listproducts.php	cat=3	Implement proper input validation and use prepared statements or parameterized queries.
Potential Cross-site Scripting (XSS)	The application does not appear to sanitize user input, which could lead to XSS vulnerabilities.	/listproducts.php	searchFor parameter in the search form	Sanitize and encode all user inputs before rendering them in the output. Use appropriate escaping functions for HTML and JavaScript contexts.
Potential Cross-site Request Forgery (CSRF)	The application does not appear to implement any CSRF protection mechanisms, which could allow attackers to perform actions on behalf of authenticated users.	/guestbook.php	N/A	Implement CSRF tokens for all state-changing requests and validate them server-side.
Insecure Server Configuration	The server is running an outdated version of PHP (5.6.40) which may contain known vulnerabilities. Additionally, the use of HTTP instead of HTTPS could expose data in transit to interception.	/listproducts.php	N/A	Upgrade to a more recent and supported version of PHP. Enforce HTTPS by obtaining an SSL/TLS certificate and redirecting all traffic from HTTP to HTTPS.

Check known CVEs for network services' versions

The screenshot displays the configuration interface of a web application security tool. On the left is a dark sidebar with navigation links: Scan History, Products, Static Code Analysis, Web App Scanning, Infrastructure Checks (highlighted with a red box), Upload Report, My Rules, Credentials, Vuln DB, and Logout. The main content area contains several configuration sections. At the top, there are dropdowns for 'Run or save' (set to 'Run now') and 'Time' (set to '00:30'). Below these is a 'Product / Eng ID' dropdown set to 'Test 2 (3)'. A red box highlights two adjacent dropdowns: 'Ping hosts' (set to 'No') and 'Scan ports' (set to 'Top 1000 TCP/UDP'). Further down, a 'Select Scan Type' dropdown is set to 'Vulnerability Scan'. Under the 'Choose scanners:' section, there are several toggle switches: OpenVAS, Nessus, Run nmap pre-scan, Nuclei, Nuclei (My Rules), Bruteforcer, and ML INFRA (highlighted with a red box and set to 'on'). A 'Submit' button is located at the bottom right of the configuration area.

Run or save: Run now

Time: 00:30

Product / Eng ID: Test 2 (3)

Ping hosts: No

Scan ports: Top 1000 TCP/UDP

Select Scan Type: Vulnerability Scan

Choose scanners:

- ☒ OpenVAS
- ☒ Nessus
- ☐ Run nmap pre-scan
- ☒ Nuclei
- ☒ Nuclei (My Rules)
- ☒ Bruteforcer
- ☒ ML INFRA

Submit

ML INFRA Sample Results.

CVE name	CVE details	Server name and version	Host IP address	Risk level as mentioned in CVE	Recommended mitigation
CVE-2021-23017	A vulnerability in the Nginx resolver allows a 1-byte memory overwrite, potentially leading to denial of service or other impacts. This issue is triggered when the "resolver" directive is used in the configuration. Versions 0.6.18 through 1.20.0 are affected.	nginx 1.19.0	44.228.249.3	High	Upgrade to Nginx 1.20.1 or later. Remove or disable the "resolver" directive if not required.

CVE name	CVE details	Server name and version	Host IP address	Risk level as mentioned in CVE	Recommended mitigation
CVE-2018-15473	OpenSSH through 7.7 is prone to a user enumeration vulnerability allowing remote attackers to enumerate valid usernames via different response messages.	OpenSSH 7.4	194.67.204.110	Medium	Upgrade OpenSSH to version 7.8 or later, or apply vendor-supplied patches.

* Results depend on the model training date, size and quality

Scan Rules / Templates Generation

Scan History

Products

Static Analysis

Dynamic Scanning

Infrastructure Checks

Upload Report

My Rules

Credentials

Vuln DB

Logout

Type

Nuclei

Rule

New

Rule text

Path Traversal vulnerability identified in a web application's file download function.

Here's the PoC:

HTTP Request:

GET /download?file=../../etc/passwd HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0

HTTP Response:

HTTP/1.1 200 OK

Content-Type: text/plain

Content-Length: 1827

Server: Apache

root:x:0:0:root:/root:/bin/bash

...

☒ Active

Delete rule

Gen with AI

What's next?

- Shaping vulnerabilities reporting and processing
- Reducing false positives
- Report enrichment (pulling data from assets, risks and ISPF inventories, APIs)
- RAG (Retrieval-Augmented Generation)
- Training models on own data (Fine-Tuning)
- More use cases
- Upgrading hardware / models



Conclusions

- LLM provide valuable insights during all types of vulnerability scanning.
- Code scanning is more accurate, then with majority of SASTs. Report may include detailed vulnerability explanation along with corrected / safe code or other practical recommendations.
- Report / text generation capabilities improving writing speed and quality.
- Enormous amount of use cases besides vulnerability analysis.
- “Production” usage requires high speed graphics cards with enough VRAM (96GB+) to support big models (80B) with sufficient context (100K+).

Questions?



Write 3 sarcastic questions from audience to person just finished presentation about LLM capabilities.

2025/04/11 09:35:57



► Thought for a second

1. "Wow, your LLM sounds so reliable! So, when will it finally learn that the Earth isn't flat? Asking for a friend who's still confused from 2019."
(Mocking outdated training data or persistent myths.)
2. "Intriguing! Could it also time-travel to fact-check itself? Or is that where the 'hallucinations' come in?"
(Sarcastically referencing LLMs' tendency to invent 'plausible' nonsense.)
3. "Fantastic! So, can it at least write a decent apology for all the times it lied to me about 'definitely knowing' something? Or is that too *meta* for its capabilities?"
(Roasting overconfidence in incorrect answers.)

2025/04/11 09:38:32