

Muhammad Firdaus Amran

- Security Analyst (Offensive)

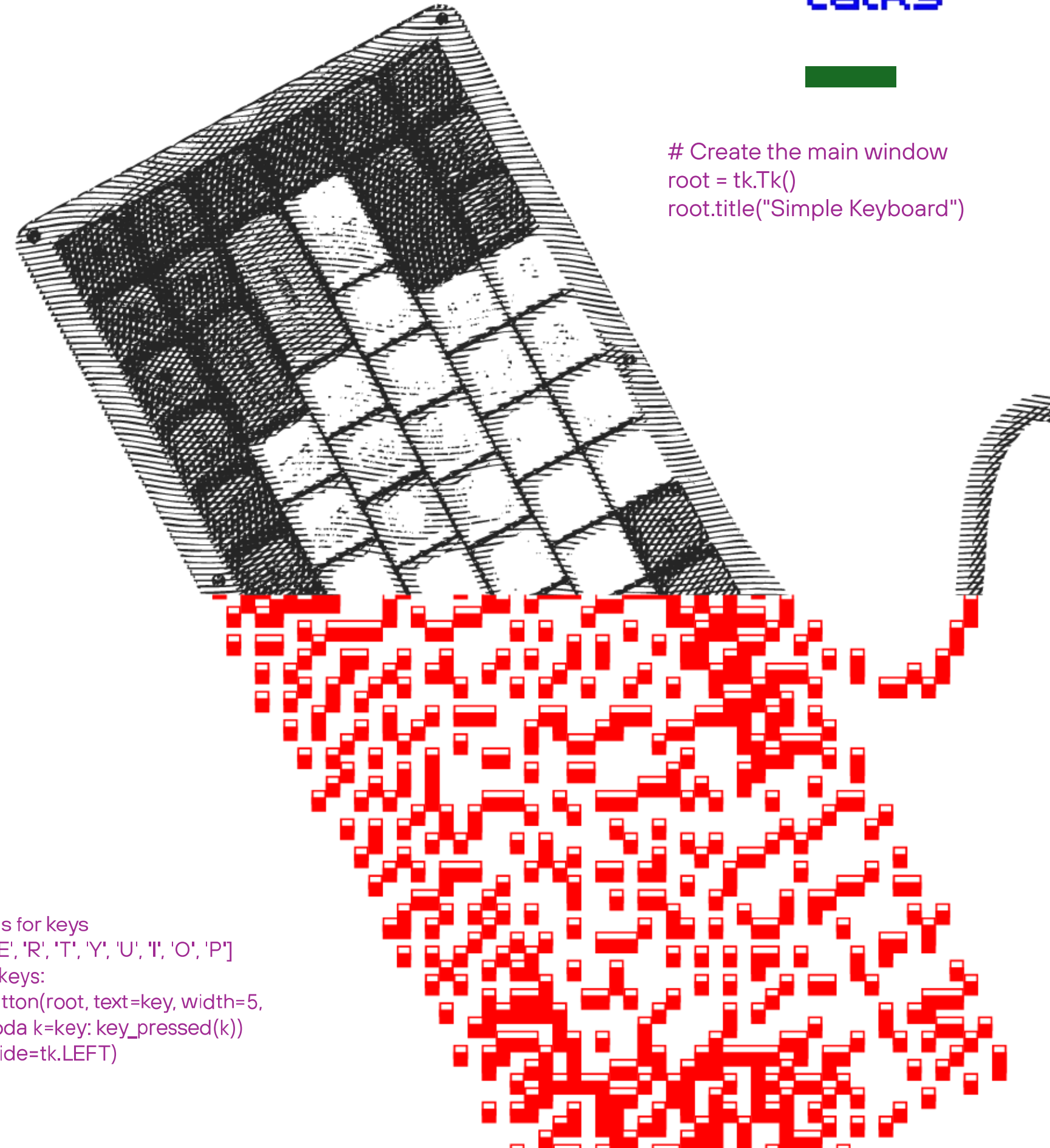
```
def key_pressed(key):  
    print(f"{key} pressed")
```

```
# Create the main window  
root = tk.Tk()  
root.title("Simple Keyboard")
```

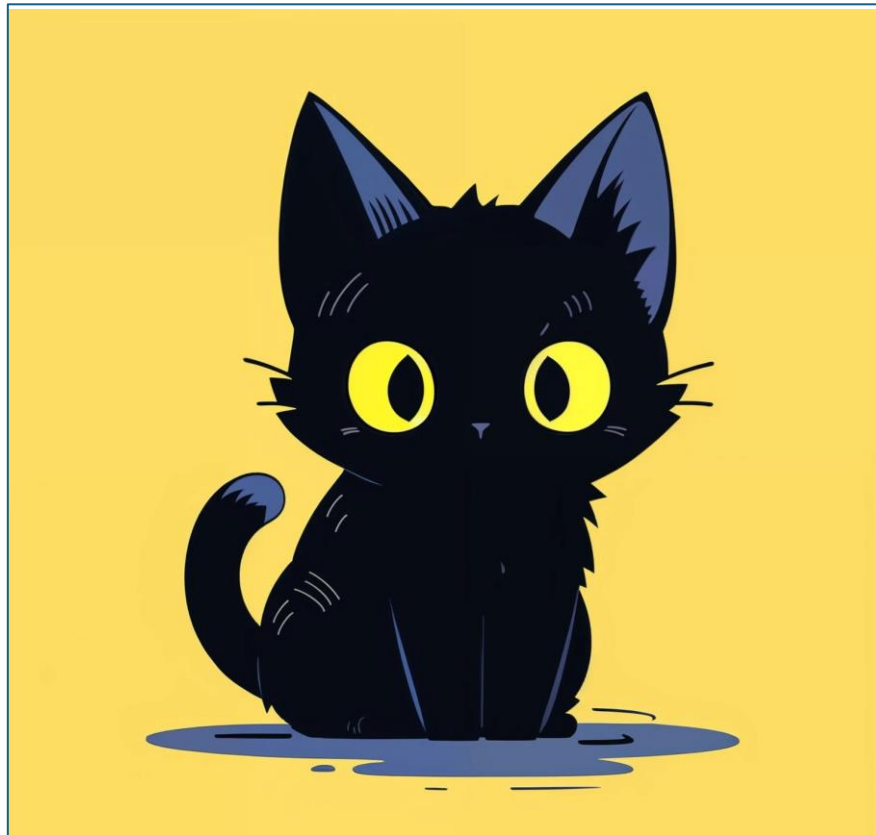
Spray, Pray, Validate: Lessons from AI-assisted Vulnerability Research

#PHTalks

```
# Create buttons for keys  
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']  
in the following keys:  
    button = tk.Button(root, text=key, width=5,  
command=lambda k=key: key_pressed(k))  
    button.pack(side=tk.LEFT)
```



\$ whoami



Muhammad Firdaus Amran (vicevirus)

- Security Analyst (Offensive)
- Former Full-stack web developer; avid CTF player and bug hunter
- Full-time analyst, part-time token gambler



Disclaimer

- This research was independently **self-funded** and carried out during my **personal time**, outside of my professional responsibilities. As a result, some areas may remain unexplored due to limitations in time, resources, and scope
- **Models used:** GPT o3, GPT-5 through GPT-5.3

Spray, Pray, Validate



Spray

Generate many vulnerability hypotheses with LLMs.



Pray

Accept that most will be noisy, but some may become useful leads.



Validate

Filter with lightweight LLM checks, then manually confirm real exploitability.

Why spray?

- Think of codebases are **huge search spaces**
- LLMs help automate the process of **exhausting more of that space**. (plus they don't get tired)
- The goal is not **perfect accuracy**, but **generating enough useful leads** to validate
- In a way, it's similar to **fuzzing**?
- Cost-wise, this approach is expensive and token-heavy :'(

Why **validate manually**?

- Minimize noisy or low-quality reports. **PoC or GTFO** (keeps the devs happy)
- Some exploit paths still need **human intuition and knowledge**
- Keep yourself in the loop. **AI should assist, not decide**

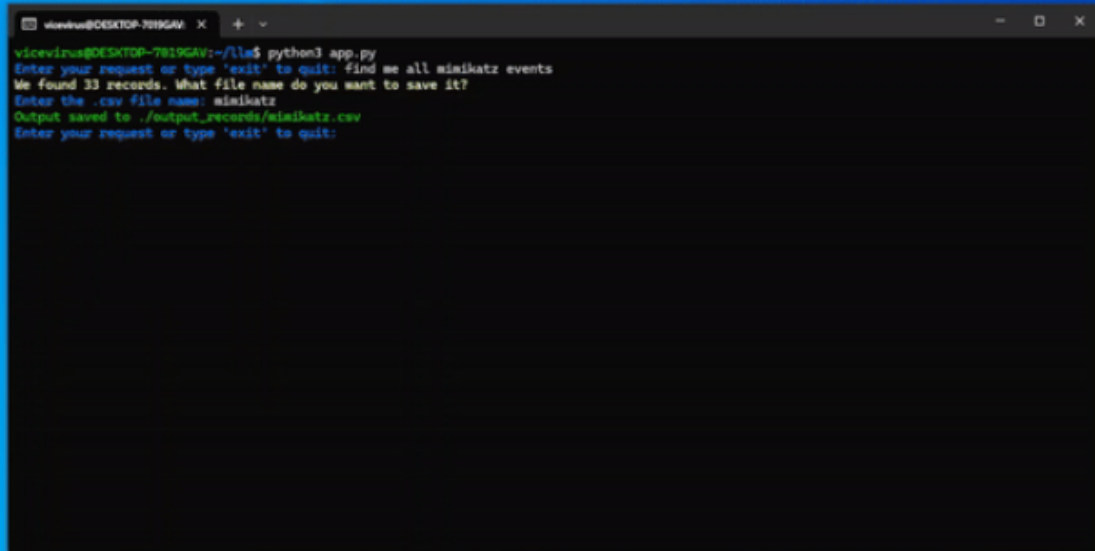
Started my journey with LLM

- Built a simple pet project chatbot with **PandasAI** and **LlamaIndex** to query Hayabusa output (forensics timeline generator) with **GPT-4** (2024)
- “Hey this LLM thing is pretty good at **querying/searching text?**”
- Of course, at this time, the model still **isn't very good at understanding and writing code** as the models we have now (2026)

LLM-Hayabusa-Logs-Parser

This is a simple script using [PandasAI](#) that allows you to easily query your Hayabusa logs with LLM such as Gemini, OpenAI etc.

Demo



```
vince@DESKTOP-7819GAV: ~$ python3 app.py
Enter your request or type 'exit' to quit: find me all minikatz events
We found 33 records. What file name do you want to save it?
Enter the .csv file name: minikatz
Output saved to ./output_records/minikatz.csv
Enter your request or type 'exit' to quit:
```

Note

This works better in GPT-4 in my experience.

Eventually models start to get better..

- Models became much better at **understanding and writing code**
- The release of **GPT-o3** pulled me back into building LLM applications again
- **GPT-o3** felt like a strong enough model to build with

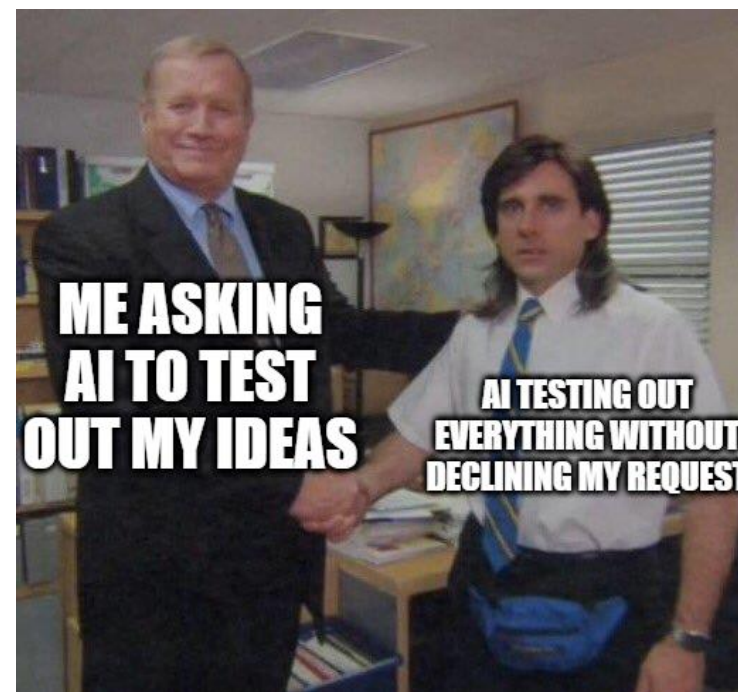


Single agent

- The main goal at the time of building a single agent is to help me **save time on CTF challenges**
- Built on **LangChain**

Single agent for ...

CTF	Vulnerability Research
For CTFs, the single-agent methodology was simple: • Throw every possible idea/hypotheses I had at the challenge • Let the agent explore what works, because it doesn't get tired (which is pretty cool!) • Guide it properly, and intervene manually when needed	• After seeing some success with CTF challenges, I wanted to try the same approach on real codebases • Built a single-agent setup for vulnerability research and had a few early successes :) • The implementation was pretty naive at the time, but it was enough to show that the idea had potential



Codeigniter 4 (CVE-2025-54418)

- Command Injection Vulnerability in ImageMagick Handler
- **CVSS 3.1:** 9.8 / Critical

Command Injection Vulnerability in ImageMagick Handler

Published

Critical

paulbalandan published GHSA-9952-gv64-x94c on Jul 26, 2025 · 2 comments

Package

Affected versions

Patched versions

php **codeigniter4/framework** ([Composer](#))

< 4.6.1

4.6.2

michalsn opened on Jul 17, 2025 · edited

Description

Impact

This vulnerability affects applications that:

- Use the ImageMagick handler for image processing (`imagick` as the image library)
- AND either:
 - Allow file uploads with user-controlled filenames and process uploaded images using the `resize()` method
 - OR use the `text()` method with user-controlled text content or options

An attacker can:

- Upload a file with a malicious filename containing shell metacharacters that get executed when the image is processed
- OR provide malicious text content or options that get executed when adding text to images

Patches

Workarounds

Upgrade to v4.6.2 or later.

- Switch to the GD image handler (`gd`, the default handler), which is not affected by either vulnerability
- For file upload scenarios: Instead of using user-provided filenames, generate random names to eliminate the attack vector with `getRandomName()` when using the `move()` method, or use the `store()` method, which automatically generates safe filenames
- For text operations: If you must use ImageMagick with user-controlled text, sanitize the input to only allow safe characters: `preg_replace('/[^\a-zA-Z0-9\s.,!?-]/', '', $text)` and validate/restrict text options

Severity

Critical 9.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVE ID

CVE-2025-54418

Weaknesses

► CWE-78

Credits

vicevirus

Reporter ✓

Collaborators

Only the following users and teams can see and collaborate on this advisory:

Codeigniter 4 (CVE-2025-54418)

- ***process()*** builds an ImageMagick command through string concatenation.
- A filename can enter the image resize flow and become part of the shell command
- Filename sanitization is not shell escaping, so shell metacharacters may still reach ***exec()***

```
$cmd = $this->config->libraryPath;  
// starts building shell command  
  
$cmd .= $action === '-version'  
    ? ' ' . $action  
    : ' -quality ' . $quality . ' ' . $action;  
// user-controlled filename/action is concatenated here  
  
@exec($cmd, $output, $retval);  
// command is executed by the shell
```

Octo-STS (CVE-2025-52477)

- Unauthenticated SSRF with HTTP Response Reflection in OIDC Flow
- **CVSS 3.1: 8.6 / High**
- Relatively small codebase

Unauthenticated SSRF with HTTP Response Reflection in OIDC Flow

Published High cpanato published GHSA-h3qp-hwvr-9xcq on Jun 27, 2025 · 16 comments

Package	Affected versions	Patched versions
github.com/octo-sts/app (Go)	<= v0.5.2	v0.5.3

Severity High 8.6 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	None
Availability	None

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

CVE ID
CVE-2025-52477

Weaknesses
► CWE-918

Credits

vicevirus	Reporter	✓
cpanato	Remediation developer	✓
mgreau	Remediation developer	✓
eslrm	Coordinator	✓
cmdpdx	Remediation reviewer	●

vicevirus opened on Jun 13, 2025 · edited by eslrm

Description

Summary

Octo-STS versions before v0.5.3 are vulnerable to unauthenticated SSRF by abusing fields in OpenID Connect tokens. Malicious tokens were shown to trigger internal network requests which could reflect error logs with sensitive information.

Please upgrade to v0.5.3 to resolve this issue. This version includes patch sets to [sanitize input](#) and [redact logging](#).

Many thanks to [@vicevirus](#) for reporting this issue and for assisting with remediation review.

References

- [GHSA-h3qp-hwvr-9xcq](#)
- [b3976e3](#)
- [0f177fd](#)

vicevirus added themselves as a collaborator on Jun 13, 2025
 vicevirus was credited as a reporter on Jun 13, 2025
 vicevirus accepted credit on Jun 13, 2025

[Decline credit](#)

Moodle (CVE-2026-26046)

- Post-auth Command Injection in TeX filter settings
- **CVSS 3.1: 7.2 / High**
- Big codebase, lucky?



MSA-26-0002: Remote code execution risk in TeX filter admin setting

by Michael Hawkins - Tuesday, 17 February 2026, 6:30 PM

Additional sanitizing was required on a TeX filter administration setting to prevent a remote code execution risk.

Note: The affected setting could only be accessed by site administrators, and only affected sites with the TeX notation filter enabled and ImageMagick installed on the server.

Severity/Risk:	Serious
Versions affected:	5.1 to 5.1.1, 5.0 to 5.0.4, 4.5 to 4.5.8 and earlier unsupported versions
Versions fixed:	5.1.2, 5.0.5 and 4.5.9
Reported by:	Vicevirus
CVE identifier:	CVE-2026-26046
Changes (main):	http://git.moodle.org/gw?p=moodle.git&a=search&h=HEAD&st=commit&s=MDL-87843 http://git.moodle.org/gw?p=moodle.git&a=search&h=HEAD&st=commit&s=MDL-87870
Tracker issue:	MDL-87843 and MDL-87870 Remote code execution risk in TeX filter admin setting

[Permalink](#)

[Discuss this topic](#) (0 replies so far)

Takeaways from the single agent setup



Agent performance gets worse as more tools are called due to **context dilution**



File reading (tool) selection can be **random**, especially in **large codebases**



The agent often **misunderstands the application threat model** and might treat any reachable sinks as vulnerabilities



Agents are **relentless**. They never get tired, so you can keep exploring the space

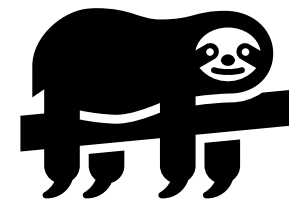


NOOOO YOU MUST
EXPLORE MANUALLY

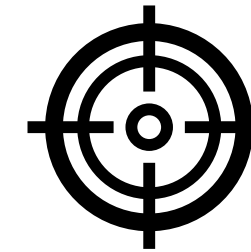


AGENT
GOES BRRRRRRR

Context dilution and performance



A single agent can run for a long time without solving the task, while **getting slower** and **less useful over time**



A single agent work best on bounded tasks with **clear objectives**, **focused instructions**, and **defined success criteria**

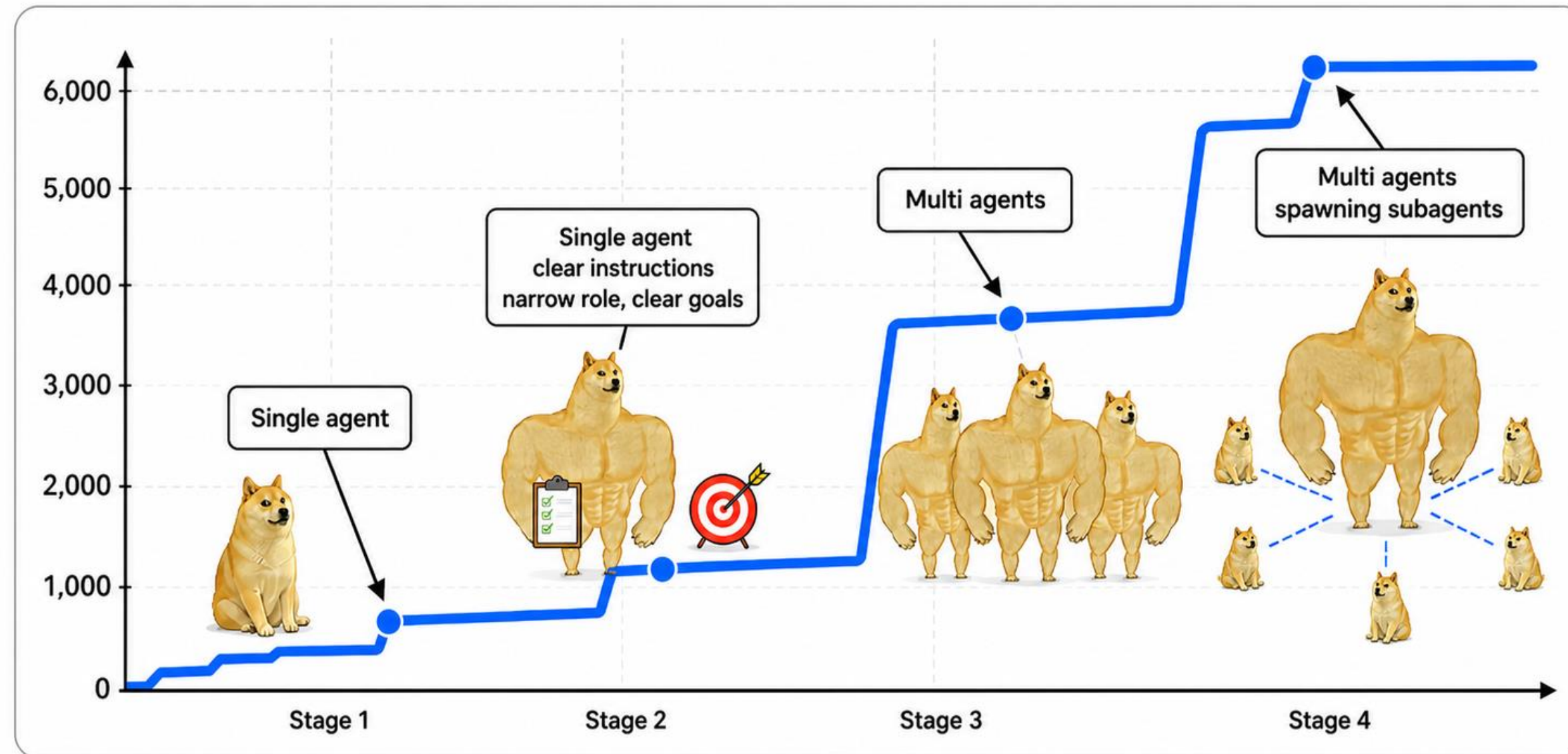


TLDR: Don't make one agent do everything. Instead, make it do one thing well

Let's make things better?

- Enter **multi-agents** and **subagents** setup
 - **multi agents** = multiple agents working together, usually with different specific roles
 - **subagents** = smaller helper agents spawned by a main agent to handle specific tasks
- **Goal:** ensure that **every agent** and **subagent** operates within a **clearly defined scope**, with **narrow responsibilities** and **clear success criteria**.

Let's make things better?



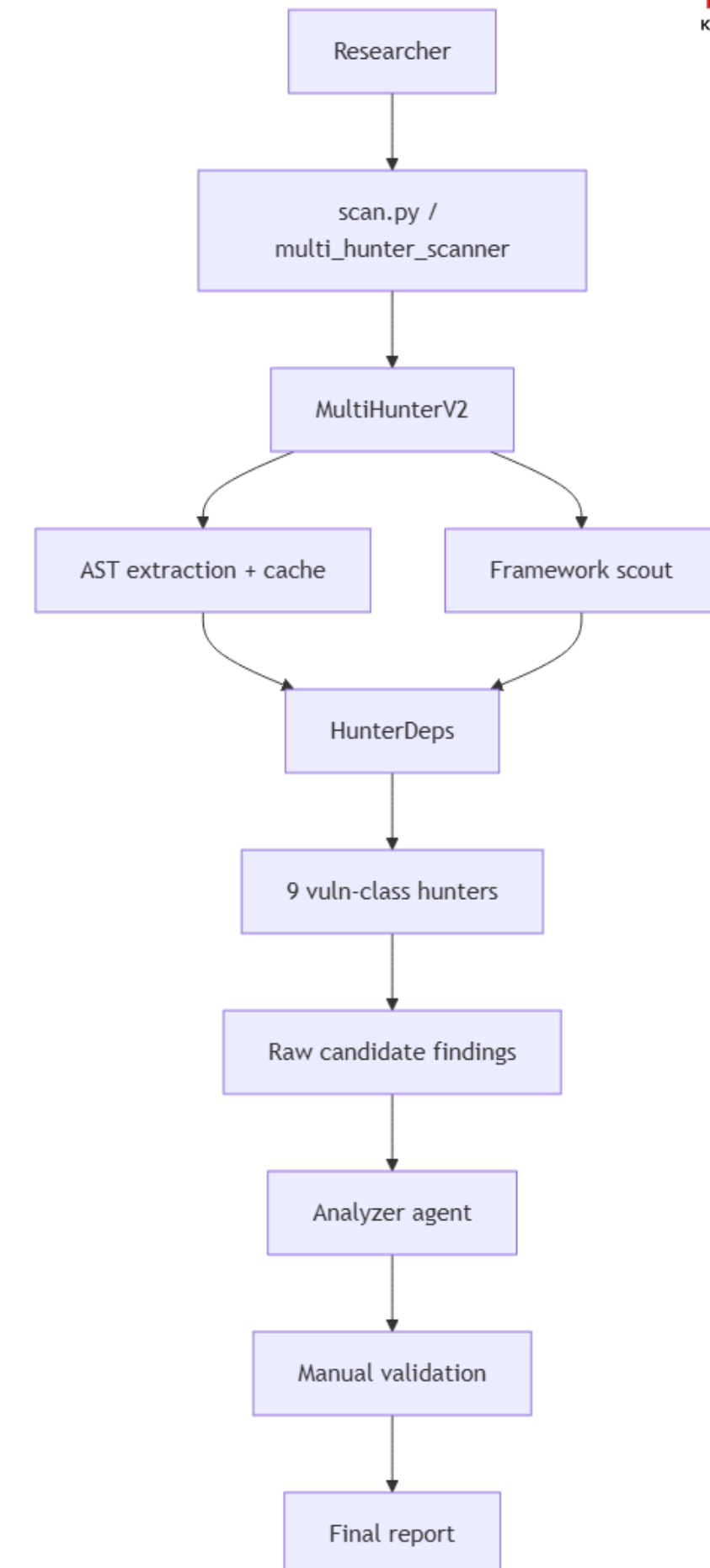
Multi-agents setup

- Tried both **PydanticAI** and **LangGraph**. Both were good and fairly similar, but **PydanticAI** felt more natural to me
- **PydanticAI** comes with structured output, context compaction, logging, skills, and more, which made building agents much easier
- **Methodology:** generate many hypotheses, validate lightly and validate manually for actual exploitability
- **Hunter agents** were mostly instructed to report **anything suspicious**, not to validate **deeply**
- **Goal:** find web-based vulnerabilities, language-agnostic

First iteration (Multi-vuln class)

- Scout agent provides **baseline context** for the other agents
- AST parsing with **treesitter** + **ripgrep** based tool
- 9 vulnerability-class hunters cover RCE, XSS, SSTI, SSRF, and more
- Generate lots of **hypotheses** from the codebase
- Analyzer agents validate **lightly**
- Validate **manually**

AST – Abstract Syntax Tree



First iteration (Chrome-vuln class)

- Scout agent provides **baseline context** for the other agents
- ripgrep-based** tool
- 5 Chrome-vuln class hunters
- Generate lots of **hypotheses** from the codebase
- Analyzer agents validate **lightly**
- Validate **manually**

Other vulnerability classes

For these classes of bugs, reports are expected to clearly demonstrate the exploitability and impact to a user, such as a convincing UI spoof or how user information would be disclosed.

Category	High Impact
UXSS Site isolation bypass	Up to \$10,000
User information disclosure	Up to \$5,000
Local privilege escalation	Up to \$5,000
Omnibox URL Spoofing	Up to \$5,000
Web platform privilege escalation	Up to \$5,000

Budibase Cloud (CVE-2026-27702)

- Remote Code Execution in View Filter Map Function
- **CVSS 3.1: 9.9 / Critical**
- The **exploit only worked on the cloud deployment**, which **increased impact** because it could expose other users' data

Remote Code Execution via Unsafe eval() in View Filter Map Function (Budibase Cloud)

Published

Critical

mjashanks published GHSA-rvhr-26g4-p2r8 on Feb 24 · 2 comments

Package	Affected versions	Patched versions	Severity
 budibase (npm)	<= 3.30.3	None	Critical 9.9 / 10

vicevirus opened on Feb 21 · edited

Description

Summary

A critical unsafe `eval()` vulnerability in Budibase's view filtering implementation allows any authenticated user (including free tier accounts) to execute arbitrary JavaScript code on the server. **This vulnerability ONLY affects Budibase Cloud (SaaS)** - self-hosted deployments use native CouchDB views and are not vulnerable. The vulnerability exists in `packages/server/src/db/inMemoryView.ts` where user-controlled view map functions are directly evaluated without sanitization.

The primary impact comes from what lives inside the pod's environment: the `app-service` pod runs with **secrets baked into its environment variables**, including `INTERNAL_API_KEY`, `JWT_SECRET`, CouchDB admin credentials, AWS keys, and more. Using the extracted CouchDB credentials, we verified direct database access, enumerated all tenant databases, and confirmed that user records (email addresses) are readable.

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	High
Availability	Low

[Learn more about base metrics](#)

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:L

CVE ID

CVE-2026-27702

Budibase Cloud (CVE-2026-27702)

- User-controlled view filter could reach *eval()* through the generated map function
- This led to **server-side code execution** in the Budibase Cloud app-service context

```
"HOSTNAME": "app-service-5f4f6d796d-p6dhz",
"NODE_VERSION": "22.22.0",
"YARN_VERSION": "1.22.22",
"PORT": "4002",
"COUCH_DB_URL": "http://admin:[REDACTED]@internal-chesterfield-prod-alb-961152818.eu-west-1.elb.amazonaws.com.:5984",
"BUDIBASE_ENVIRONMENT": "PROD",
"SERVICE": "app-service",
"POSTHOG_TOKEN": "phc_[REDACTED]",
"ACCOUNT_PORTAL_URL": "https://account.budibase.app",
"TOP_LEVEL_PATH": "/app",
"BUDIBASE_VERSION": "3.30.3",
"DD_GIT_REPOSITORY_URL": "https://github.com/budibase/budibase",
"DD_GIT_COMMIT_SHA": "668435f9e278f402a78e0536b7e9d1b7264e997c",
"DD_VERSION": "3.30.3",
"NODE_ENV": "production",
"NODE_OPTIONS": "--no-node-snapshot --require /opt/datadog/apm/library/js/node_modules/dd-trace/init.js",
"CLUSTER_MODE": "",
"CLOUDFRONT_PUBLIC_KEY_ID": "KL23NAZBW7UD1",
"MINIO_URL": "",
"JS_PER_REQUEST_TIME_LIMIT_MS": "60000",
"DD_INTERNAL_POD_UID": "12cab32a-3406-4aca-9646-cebbf486cba7",
"JWT_SECRET": "A9D8[REDACTED]",
"MINIO_ENABLED": "false",
"MINIO_ACCESS_KEY": "AKI[REDACTED]",
"WORKER_URL": "http://worker-service:4003",
"DD_DOGSTATSD_URL": "unix:///var/run/datadog/dsd.socket",
"CLOUDFRONT_CDN": "https://d2l5prqdbvm3op.cloudfront.net",
```

Chrome Browser (CVE-2026-3938)

- Insufficient policy enforcement in Clipboard in Google Chrome
- **CVSS 3.1: 4.3 / Medium**

🦟 CVE-2026-3938 Detail

Description

Insufficient policy enforcement in Clipboard in Google Chrome prior to 146.0.7680.71 allowed a remote attacker who had compromised the renderer process to leak cross-origin data via a crafted HTML page. (Chromium security severity: Low)

Metrics

CVSS Version 4.0

CVSS Version 3.x

CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:



NIST: NVD

Base Score: **4.3 MEDIUM**

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N

ADP: CISA-ADP

Base Score: **6.5 MEDIUM**

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N

[\$2000][474763968] **Low** CVE-2026-3938: Insufficient policy enforcement in Clipboard. Reported by vicevirus on 2026-01-10

Takeaways from the first iteration

- Multi-agent and subagent strategies **worked well, up to a point**
- Different roles helped, but **large codebases still overwhelmed individual agents**, and their exploration could still **drift unpredictably**
- AST parsing was **useful for structure**, but cross-file tracking across languages became **messy** due to language-specific imports, framework patterns, and dataflow behavior

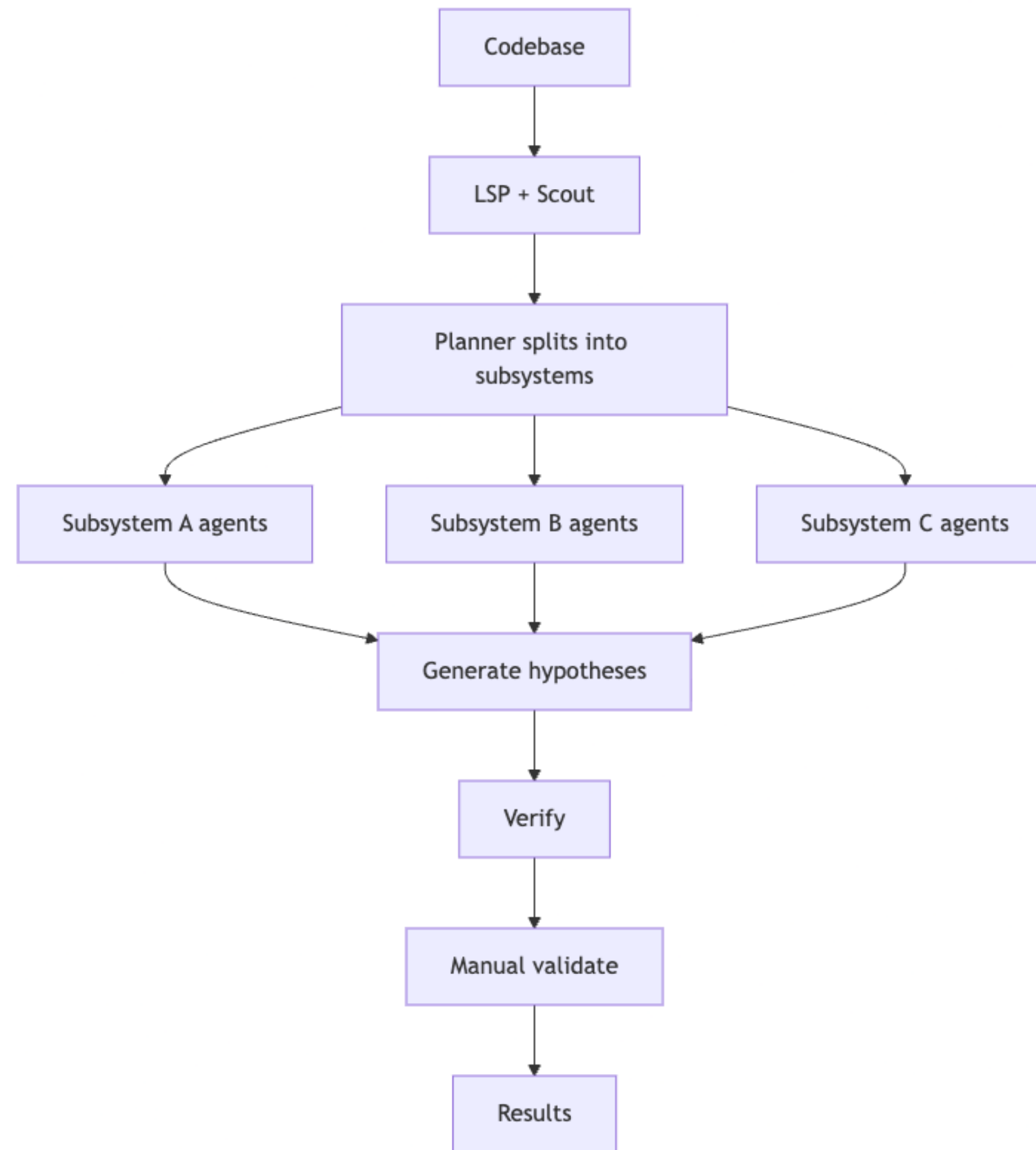
Making things even better? (also expensive)

- Scrapped **AST parsing**, relied on **ripgrep-based tooling** instead
- Added **LSP support** for most languages
- Added a **planner agent** to **split the codebase into smaller subsystems**
- Assigned **multiple agents to each subsystem** so they could generate more hypotheses **within a smaller scope**
- **Why:** smaller scopes reduce random file-reading and produce more focused hypotheses

GENERATE EVEN MORE HYPOTHESES!!!

LSP– Language Server Protocol

Making things even better? (also expensive)



Final takeaways

- 1 Use LLMs like a **tireless** fuzzing engine to **exhaust the search space**
- 2 For agents and subagents: **small task, small scope, simple tools**
- 3 Avoid being too **language/project-agnostic**. You will mostly surface shallow bugs
- 4 Build **codebase-specific harnesses** when possible especially if you have RFCs, docs and specs
- 5 **Harness design >>> everything else, even fine-tuning**
- 6 LLMs will improve as they get better at **reading, writing, and reasoning** about code

Harness = codebase-specific setup: tools, prompts, docs,
and checks that guide agents

Future work....

- Many generated findings were still not fully validated
- Some were validated, but not submitted yet, still under embargo, or had no response from maintainers
- The remaining leads are left for future work :))

Question

Positive Hack Talks

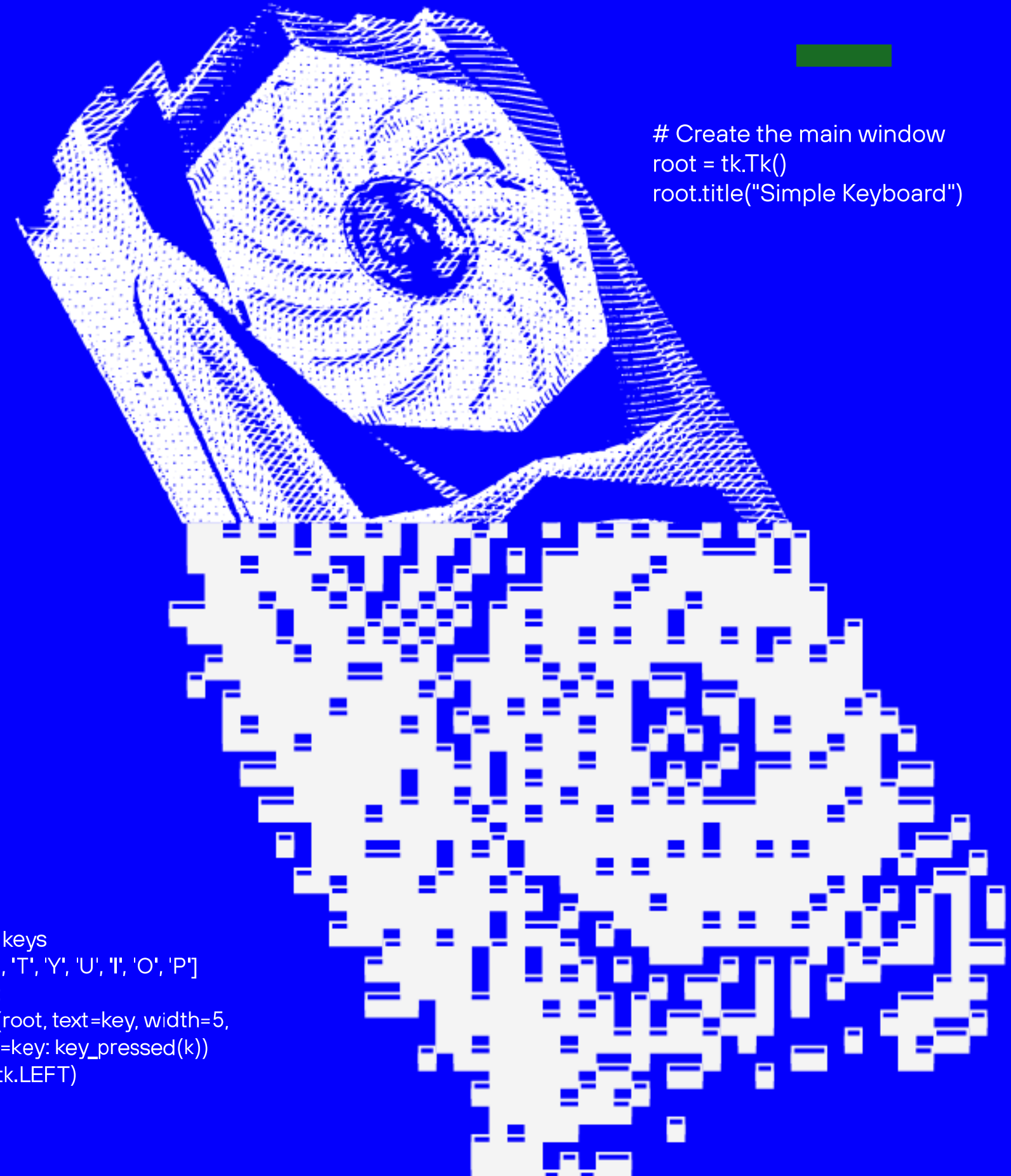


```
def key_pressed(key):  
    print(f"{key} pressed")
```

```
# Create the main window  
root = tk.Tk()  
root.title("Simple Keyboard")
```

#PHTalks

```
# Create buttons for keys  
keys = ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P']  
in the following keys:  
    button = tk.Button(root, text=key, width=5,  
command=lambda k=key: key_pressed(k))  
    button.pack(side=tk.LEFT)
```



positive Kuala Lumpur hack talks

by ■ positive technologies

```
# Display the cube and voxel  
count  
fig, ax = plt.subplots  
(subplot_kw={"projection": "3d"})  
ax.voxels(cube, edgecolor='k')  
ax.set_title(f"Cube with  
{np.count_nonzero(voxels)}  
voxels")  
plt.show()
```

```
import numpy as np  
import matplotlib.pyplot as plt
```

Thank you

#PHTalks