



[AI FOR ETHICAL HACKING]

root@ethiack:/# whoami

André Baptista | @0xacb

[> WORKSHOP]
[VERSION] → 1.0
[DATE] → June 25, 2025

Contents

1	Introduction	5
1.1	Current Context	5
1.2	Vibe coding risks	5
1.3	An opportunity	11
1.4	Responsible Use and Legal Considerations	11
2	Setup	12
2.1	Required software	12
2.2	Gemini free tier API key	12
2.3	LLM tool	12
3	Reconnaissance	13
3.1	Subdomain Enumeration	13
3.1.1	Prompt-based	13
3.1.2	Subwiz	13
3.1.3	Contextual wordlists	14
3.2	Screenshot Analysis	17
3.2.1	Eyeballer	17
3.2.2	Multimodal LLMs	18
3.3	Content Discovery	20
3.3.1	FFUFAI	20
3.3.2	Contextual wordlists	21
4	Exploit Generation	23
4.1	Nuclei AI flag	23
4.2	One-day exploit generation	24
5	Hackbots	25
5.1	CAI	25
5.1.1	Solving CTF challenges	25
5.1.2	Extending CAI with custom tools	26
5.2	Burp AI	27
6	MCP servers	30
6.1	Burp Suite MCP	30
6.2	Other Examples	31
6.2.1	MCP for Security project	31
6.2.2	Shift Plugin	32
6.2.3	Ghidra MCP	32
7	Conclusion	33

List of Figures

1	Zero Factor Authentication (0FA) viral vibe code tweet	6
2	Real scenario showing user email exposure in Application Programming Interface (API) responses	6
3	Timeline of compromises in Harley's disclosed hacker directory	7
4	Claude tasks for simple PHP application	8
5	Uploading PHP file to the application	10
6	Arbitrary code execution through upload directory	10
7	Subwiz results for apple.com	14
8	Gowitness Gallery	17
9	Eyeballer results for anthropic.com	18
10	FFUFAI execution example	21
11	Executing the AI-generated CVE-2025-32433 exploit	24
12	CAI CLI Overview	25
13	Resulting discord notification	27
14	Deterministic Issues from Crawl and Audit	28
15	AI Task Result	28
16	AI Task Request History	29
17	MCP components overview [21]	30
18	Burp MCP example with Claude Desktop	31
19	Burp Suite MCP approval	31
20	Shift Plugin automatically setting a fuzzing-ready wordlist	32
21	Ghidra MCP available functions	32

List of Listings

1	Vulnerable PHP file upload application generated by Claude	9
2	Docker command to pull and run image with all necessary tools	12
3	Installing and configuring LLM tool with Gemini	12
4	AI-generated subdomain enumeration for Tesla	13
5	Using Subwiz for AI-powered subdomain generation	13
6	Using HTTPX and html2text to extract text from responses	14
7	NLTK Python code example	15
8	Using NLTK to extract keywords	15
9	Using PureDNS to resolve subdomains with the combined wordlist	16
10	ScrapeGraphAI script example	16
11	Combining ScrapeGraphAI with NLTK	16
12	Using Eyeballer to classify screenshots	17
13	Script example to classify screenshots with Gemini	20
14	Running custom Gemini screenshot analysis script	20
15	Installing and running ffufai	21
16	Generating fuzz wordlist from the content of the root webpage	22
17	Nuclei AI flag example for template injection	23
18	Running CAI in CTF mode	26
19	Extending CAI with tools	27

List of Acronyms

OFA Zero Factor Authentication. 2, 6

AI Artificial Intelligence. 5, 6, 11, 33

API Application Programming Interface. 2, 5, 6, 12, 18

CAI Cybersecurity AI. 25, 26, 31, 33

CTF Capture the Flag. 25

CVE Common Vulnerability and Exposure. 26

DNS Domain Name System. 13, 15

GPU Graphics processing unit. 12

HITL Human-in-the-Loop. 30

LLM Large Language Model. 5, 12–15, 18, 20, 22, 23, 33

MCP Model Context Protocol. 30–32

NLP Natural Language Processing. 14, 15

NLTK Natural Language Toolkit. 14, 16

PHP PHP: Hypertext Preprocessor. 7

PoC Proof of concept. 24, 33

RAG Retrieval-Augmented Generation. 33

SSH Secure Shell. 24

SVG Scalable Vector Graphics. 26

URL Uniform Resource Locator. 11

XSS Cross-Site Scripting. 26

1 Introduction

1.1 Current Context

Artificial Intelligence (AI). Still a buzzword, still taking time to perfect the beat. But it is actually changing the world and there is a race to deploy it. Sometimes, it is used to amplify the good. Sometimes, it is deployed recklessly, without thinking about the potential consequences.

In fact, today, we are at a tipping point or the edge of a revolution, driven by the same force that is reshaping nearly every other industry: artificial intelligence. Mostly Large Language Models, because they are the most accessible and immediately deployable AI technology we have seen to date. Unlike other AI models, they can be easily integrated with relatively simple API calls. They are not good or perfect for every task, but generic enough across a significant range of use cases.

AI is changing the way we build software. Developers have been using technologies like Copilot for a while to write code faster than ever before. Today, there are new tools that make this even easier, also known as vibe coding tools - just talk with the model and it will build everything from scratch, test it and fix it (e.g. Claude Code, Cursor) in an interactive manner. But this speed comes with a cost. We were already scaling the technology without testing it enough before. More than 25% of Google's code is now written by AI [1]. Programmers using GitHub Copilot report up to 75% higher satisfaction rates compared to programmers who do not use any type of LLM in their workflow. Additionally, these programmers are also up to 55% more productive at writing code [2]. With this wave of AI-generated code creating billions of lines of code, there is more complexity, and thus more vulnerabilities.

“Complexity is the worst enemy of security” [3]

Will technology be stable enough to survive this? Well, security needs to keep up, both on the defensive and offensive sides. There is a really nice book on how AI is transforming our information networks, not limited to technology-based networks but also democracies, which can be viewed as information networks. Nexus of Yuval Noah Harari [4]. I invite you to take a look to understand a part of what is going on.

So, the question is not whether AI will reshape our world. It already is. The question is whether we can handle this transformation, ensuring that the race to deploy does not leave human values, ethics, and safety behind. As always, humans and society will adapt. Crackers will adapt, hackers will adapt.

1.2 Vibe coding risks

While AI tools boost productivity, they may also introduce new security challenges as we are going to explore in this section. The security community is no longer looking for common human errors but instead looking for unforeseen consequences of the AI-generated code. Attack surfaces grow faster, more assets are deployed, and more applications are being deployed. I hypothesize that this will create new attack surfaces, with more custom applications instead of third-party solutions because it may be cheaper and easier to quickly ask an AI to code it and self-host it. As an example that may illustrate this hypothesis, multiple open source clones of the [Cluely](#) app appeared before and after the company even raised \$15M USD [5]. However, these applications may have vulnerabilities that may be incredibly simple and obvious to a trained eye.

Account Verification

We have just sent the code **435841** to your phone
number: xxx-xxx-8247

Please enter the code below to access your account:



Figure 1: 0FA viral vibe code tweet

This example 0FA (Figure 1) was just a viral tweet rather than a real vulnerability, but it got me thinking. It illustrates how sensitive data could potentially be exposed through unintentional channels, such as being leaked into API response objects. While this specific case appears fabricated, security researchers should audit how sensitive data flows through an application's entire stack, and look for authorization issues. Some vulnerabilities introduced by AI may be absurd, we know that models hallucinate and they lack context about authorization. This was similar to a real scenario where user emails were being exposed:



Figure 2: Real scenario showing user email exposure in API responses

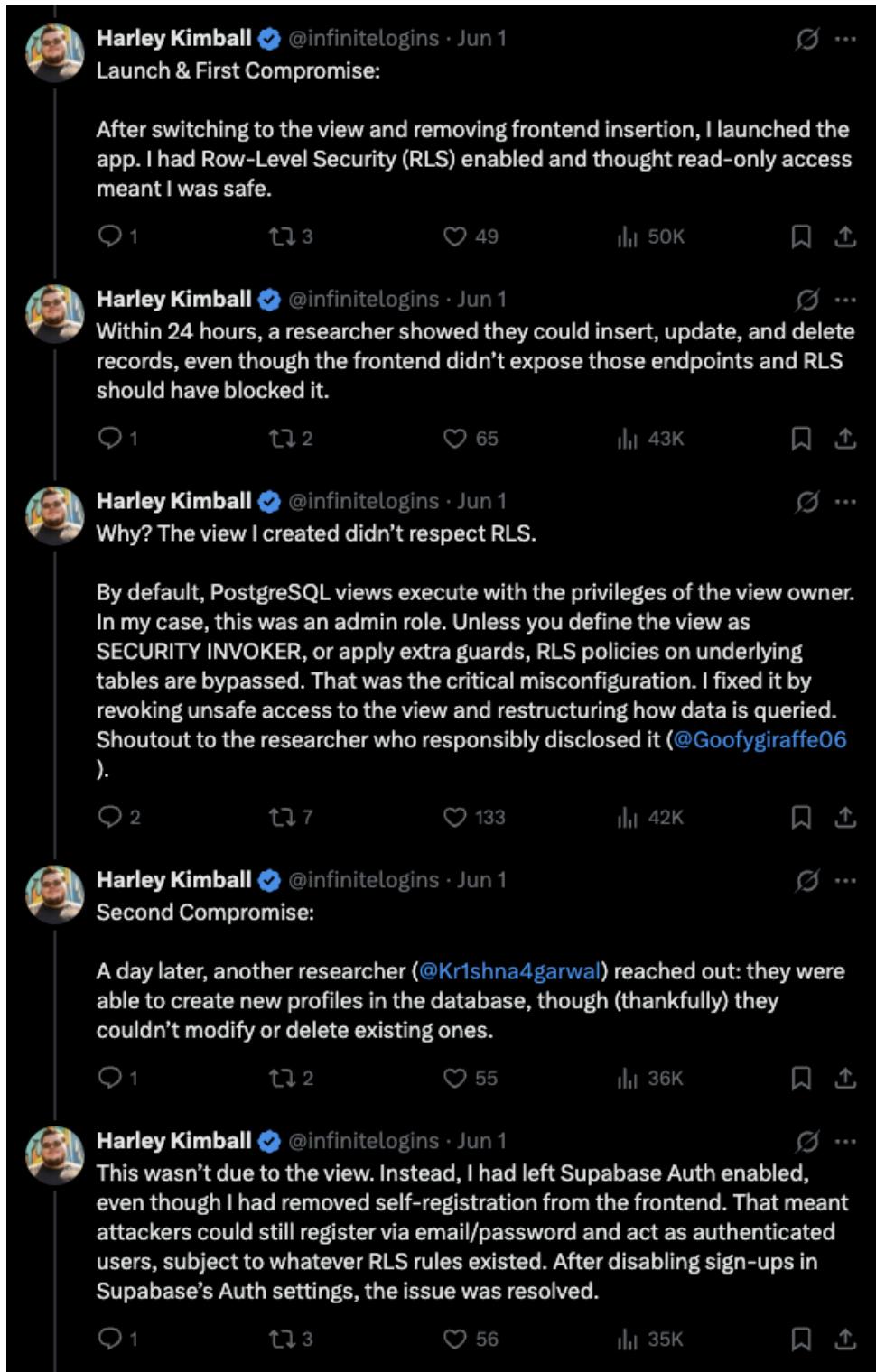


Figure 3: Timeline of compromises in Harley’s disclosed hacker directory

Harley’s disclosed hacker directory [6] was hacked twice (Figure 3). The core of the story and the reason it gained attention is that his app was hacked twice by white-hat hackers who exposed significant security vulnerabilities. Kimball mentions the dangers of prioritizing speed and “good vibes” over robust security, especially when using complex tools like Supabase and PostgreSQL.

However, classical vulnerabilities may still exist, but seem to be more rare when the best models are used and secure functions are used, e.g., `basename` in PHP: Hypertext Preprocessor

(PHP) to avoid path traversal vulnerabilities. Consider the following app generated with Claude code:

```
> create a simple php app that allows users to upload and view files
• I'll create a simple PHP file upload and viewing application. Let me plan this task first.

● Update Todos
└ □ Create main PHP file with upload form and file listing
  □ Create uploads directory for storing files
  □ Add file upload handling logic
  □ Add file viewing/download functionality

● Update Todos
└ □ Create main PHP file with upload form and file listing
  □ Create uploads directory for storing files
  □ Add file upload handling logic
  □ Add file viewing/download functionality

● Write(index.php)
└ Wrote 101 lines to index.php
  <?php
  $uploadDir = 'uploads/';
  $message = '';
  if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_FILES['file'])) {
    if (!is_dir($uploadDir)) {
      mkdir($uploadDir, 0755, true);
    }
    $file = $_FILES['file'];
    ... +91 lines (ctrl+r to expand)
  }

● Update Todos
└ ✘ Create_main_PHP_file_with_upload_form_and_file_listing
  ✘ Create_uploads_directory_for_storing_files
  ✘ Add_file_upload_handling_logic
  ✘ Add_file_viewing/download_functionality

● Created a complete PHP file upload application at index.php. The app includes:
- File upload form with drag-and-drop styling
- Automatic uploads directory creation
- File listing with download links
- Error handling and success messages
```

Figure 4: Claude tasks for simple PHP application

At this moment, please take a break from this guide and try to solve the Good Vibes challenge at ai4eh.ethiack.ninja.

Let us take a look at the following code generated by Claude code:

```
<?php
$uploadDir = 'uploads/';
$message = '';

if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_FILES['file'])) {
    if (!is_dir($uploadDir)) {
        mkdir($uploadDir, 0755, true);
    }

    $file = $_FILES['file'];
    $fileName = basename($file['name']);
    $targetPath = $uploadDir . $fileName;

    if ($file['error'] === UPLOAD_ERR_OK) {
        if (move_uploaded_file($file['tmp_name'], $targetPath)) {
            $message = "File uploaded successfully: " . htmlspecialchars($fileName);
        } else {
            $message = "Error uploading file.";
        }
    } else {
        $message = "Upload error: " . $file['error'];
    }
}

if (isset($_GET['download']) && $_GET['download']) {
    $file = $uploadDir . basename($_GET['download']);
    if (file_exists($file)) {
        header('Content-Description: File Transfer');
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment; filename=' . basename($file) . '');
        header('Expires: 0');
        header('Cache-Control: must-revalidate');
        header('Pragma: public');
        header('Content-Length: ' . filesize($file));
        readfile($file);
        exit;
    }
}

$files = [];
if (is_dir($uploadDir)) {
    $files = array_diff(scandir($uploadDir), ['.', '..']);
}
?>
```

Listing 1: Vulnerable PHP file upload application generated by Claude

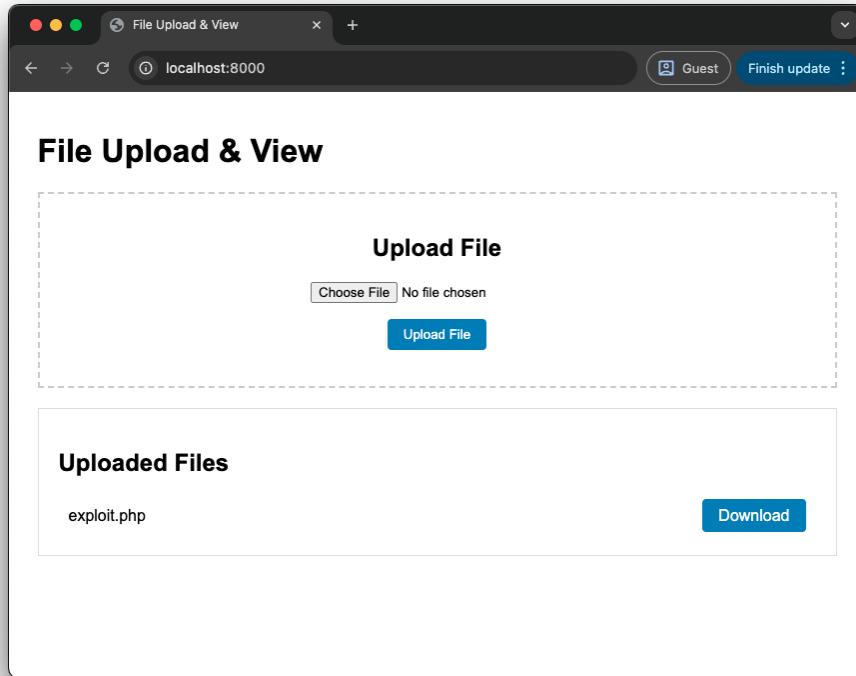


Figure 5: Uploading PHP file to the application

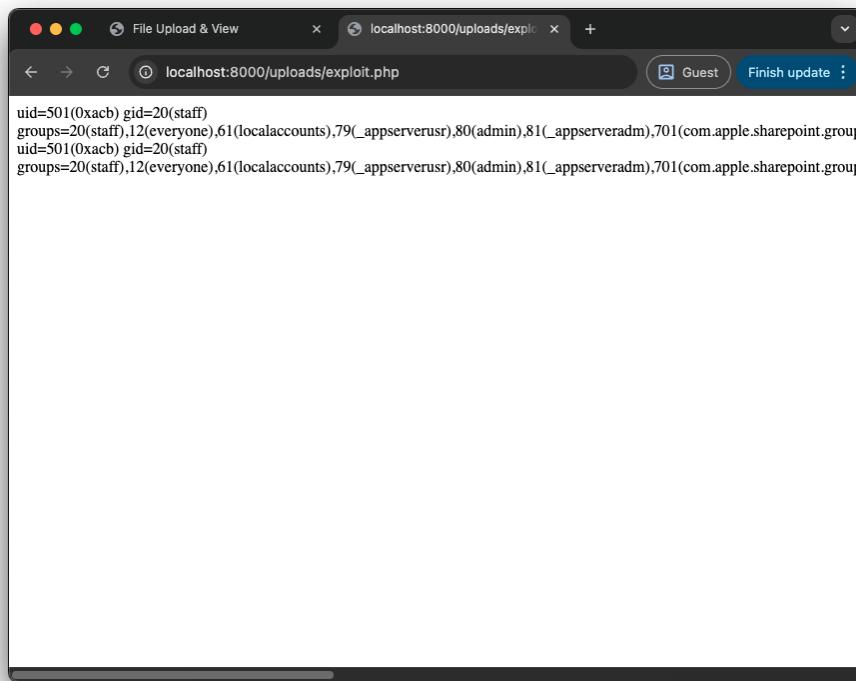


Figure 6: Arbitrary code execution through upload directory

Even if you are still learning about web security, you can easily notice that while path traversal is prevented, the upload directory can still be reached and arbitrary code will be executed (Figures

5 and 6).

1.3 An opportunity

The game has changed and the rules are being rewritten. AI tools are also accessible to those who may want to cause harm. This is an opportunity for security researchers to amplify what they can do.

In a HackerOne blog post, I mentioned: “*AI is not replacing security researchers; it is unleashing our full power. It makes us more effective in discovering and fixing vulnerabilities so that we can focus on what matters most: solving challenges that require creative thinking, collaboration, and the infinite curiosity of the human mind.*” [7]

HackerOne also mentions: “*In 2024, we saw the adoption of AI in hacking workflows take off. In a survey of over 2,000 security researchers on the HackerOne platform, 20% now see AI as an essential part of their work, up from 14% in 2023*” [8].

Penetration testing has multiple phases: from reconnaissance and subdomain enumeration, to content discovery and visual analysis, all the way through to vulnerability exploitation, and each of these phases presents opportunities for AI integration. We can generate context-sensitive wordlists for fuzzing in seconds or visually inspect thousands of URLs to identify promising targets. This is about enhancing our own skills, making us faster, more efficient, and more effective hackers. It can also be used as an autonomous agent or copilot: beyond just improving tools, we can now build and deploy hackbots, autonomous AI agents that can reason, plan and execute. These agents may complete workflows, find leads, or perform automated exploitation attempts.

In this workshop, we are going to dive into both of these areas and get hands-on with some tools. At the end of this workshop, you will not be expected to learn about how to evaluate the success of applying these techniques, which should be done through proper benchmarking, but rather you will gain some practical experience understanding when and how to effectively integrate AI into your security testing workflows.

1.4 Responsible Use and Legal Considerations

Before using any of the techniques in this workshop in the real world:

- Ensure you have authorization for all targets
- Understand the scope and limitations of your testing
- Remember that mass DNS brute forcing, automated scanning, and exploitation attempts may cause service disruption, may result in legal consequences, especially if you do not know what you are doing
- Always test tools properly in controlled environments first
- Always follow responsible disclosure practices

2 Setup

2.1 Required software

We are going to use different tools in this workshop. You may use a prebuilt Docker image.

```
$ docker run --rm -it --env-file env_file ethiack/ai4eh:latest
```

Listing 2: Docker command to pull and run image with all necessary tools

If you want to manually install them:

- python and pip - use your package manager, e.g. `apt install python3 python3-pip`
- golang - <https://go.dev/doc/install>
- subfinder - <https://github.com/projectdiscovery/subfinder>
- httpx - <https://github.com/projectdiscovery/httpx>
- puredns - <https://github.com/d3mondev/puredns>
- ffuf - <https://github.com/ffuf/ffuf>
- nuclei - <https://github.com/projectdiscovery/nuclei>
- notify - <https://github.com/projectdiscovery/notify>
- jq - <https://github.com/jqlang/jq>

You may also want to create a virtualenv to install further Python packages via pip with `pipenv`. For go-based tools, you may need to adjust your `.bashrc`, `.zshrc` or similar: `export PATH =~/go/bin:$PATH`

2.2 Gemini free tier API key

We will be using a Gemini free tier key for this workshop, but you may also use paid keys for Gemini or other providers if you have access. Alternatively, you may also install [Ollama](#) and download a model, such as `qwen3:latest`, to run locally if you have a decent GPU. Go to <https://aistudio.google.com/> [9] and follow the instructions. Accept the terms and create an API key in a new project.

2.3 LLM tool

The LLM tool is a command-line application that brings LLMs to your terminal. We can pipe the output directly into files instead of copying them from our browser. Follow the instructions: <https://llm.datasette.io/en/stable/setup.html> [10] and install the plugin for API that you are using.

```
$ llm install llm-gemini
$ llm keys set gemini
Enter key:
$ llm -m gemini-2.0-flash "hello"
Hello! How can I help you today?
```

Listing 3: Installing and configuring LLM tool with Gemini

3 Reconnaissance

3.1 Subdomain Enumeration

LLMs can be useful for generating wordlists that can be supplied to DNS brute force or to enrich permutations, for example to be used with the alterx tool, a fast and customizable subdomain wordlist generator. Let us take a look at some methods.

3.1.1 Prompt-based

Prompt engineering is very important in reducing hallucinations and optimizing efficiency. At the same time, depending on the scenario, providing explicit examples to the LLM significantly improves the quality of the output, especially for complex tasks. A simple generic approach that works to create word lists for the enumeration of Domain Name System (DNS) is to use any LLM with a prompt.

```
$ llm -m gemini-2.0-flash "Create a subdomain enumeration list for Tesla. Identify Tesla's DevOp
    ↳ stack, location, and environments. Consider them all when creating the list. Avoid
    ↳ common subdomains that would exist in other security wordlists. Make the list 1000 lines
    ↳ long. Remove duplicates. Make the list easy to copy and provide only the subdomain
    ↳ names, not the full domain. Do not number the list."
    autopilot-dev
    autopilot-qa
    autopilot-staging
    gigafactory-nv
    gigafactory-tx
    gigafactory-berlin
    gigafactory-shanghai
    battery-dev
    battery-qa
    battery-staging
    energy-dev
    energy-qa
    energy-staging
    solar-dev
    solar-qa
    ...
    ..."
```

Listing 4: AI-generated subdomain enumeration for Tesla

In this case, LLMs may have been trained with Tesla and more generic infrastructure data, so they may have enough context about the target.

3.1.2 Subwiz

A [tool by Hadrian](#) that generates more subdomains based on the known subdomains of a target. Subwiz predicts the most likely sequences using a beam search algorithm and a specialized nanoGPT-based model fine-tuned for sequence prediction tasks, making it more efficient than general-purpose LLMs for this use case. The model is publicly available on Hugging Face: <https://huggingface.co/HadrianSecurity/subwiz> [11].

```
$ pip3 install subwiz
subfinder -d apple.com -o subdomains.txt && subwiz -i subdomains.txt -o resolved.txt
```

Listing 5: Using Subwiz for AI-powered subdomain generation

```
rjvt.apple.com
vendor-mr11.ls.apple.com
mr.cm.ireporter.apple.com
st14p01sa-sailfish-stg.siri.apple.com
[INF] Found 3141 subdomains for apple.com in 1 second 767 milliseconds

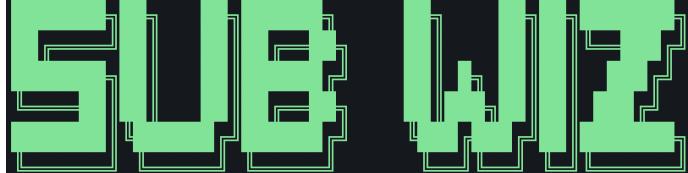

running inference.....
resolving subdomains...
→ ~ cat resolved.txt
access.apple.com
av.apple.com
c.apple.com
ext.apple.com
k.apple.com
pr.apple.com
web-uat.apple.com%
```

Figure 7: Subwiz results for apple.com

3.1.3 Contextual wordlists

What if we are performing reconnaissance on a specific target, for example, organization with non-English subdomains? We could ask an LLM to produce highly specific subdomain enumeration wordlists for these cases by providing context about the target.

We can create a tool that fetches information from the target, generates such a wordlist, and checks if the subdomains are valid.

```
$ pip3 install html2text
$ subfinder -d example.com -o subs.txt
$ echo example.com >> subs.txt # we also want to fetch main domain
$ httpx -l subs.txt -json -include-response -follow-redirects -o responses.json
$ cat responses.json | jq -r '.body' responses.json | html2text > content.txt
```

Listing 6: Using HTTPX and html2text to extract text from responses

Although this content could be provided as a context for an LLM, let us explore Natural Language Processing (NLP) techniques to extract words with the Natural Language Toolkit (NLTK). Sometimes, you do not need a machine gun to fight off a gnat. Traditional NLP methods are more efficient and cost effective for specific tasks, while LLMs offer superior generalization and contextual understanding but require significantly more computational resources. The choice depends on whether you prioritize efficiency and control, or flexibility and broad language understanding.

```
import nltk
import sys
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from collections import Counter
import re

def extract_keywords(filename, min_length=3, max_length=30):
    with open(filename, 'r', encoding='utf-8', errors='ignore') as f:
```

```

text = f.read()

text = text.lower()
text = re.sub(r'[^w\s]', ' ', text)

tokens = word_tokenize(text)

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

keywords = []
for token in tokens:
    if (len(token) >= min_length and
        len(token) <= max_length and
        token not in stop_words and
        not token.isdigit() and
        re.match(r'^[a-zA-Z][a-zA-Z0-9]*$', token)):

        lemmatized = lemmatizer.lemmatize(token)
        keywords.append(lemmatized)

word_freq = Counter(keywords)

unique_keywords = sorted(word_freq.keys(), key=lambda x: (-word_freq[x], x))

return unique_keywords

if __name__ == "__main__":
    keywords = extract_keywords(sys.argv[1])

    for keyword in keywords:
        print(keyword)

```

Listing 7: NLTK Python code example

```

$ pip3 install nltk
>>> python3
import nltk
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
$ python3 nlp.py content.txt > extracted.txt

```

Listing 8: Using NLTK to extract keywords

Then, you can use `puredns` or a similar tool to perform DNS brute force with this list without using an LLM. However, it may still be interesting to try to input these keywords into an LLM to enrich it with similar words. It is also important to mention that sometimes we cannot provide huge content to LLMs or we may want to save tokens, so NLP in general can help reduce input to what matters most. Please note that performing DNS brute force at scale needs to be an authorized activity by the target, as it can result in potential disruptions.

```

$ echo "Based on the following keywords extracted from a target, generate a big, unique and
    ↵ clever subdomain wordlist related with those. It must not contain spaces but it may
    ↵ contain dashes. Return only the wordlist without any domain and nothing else: $(cat
    ↵ extracted.txt)" | llm -m gemini-2.0-flash > llm_subs.txt

plano-higiene
planohigiene

```

```

segurança-alimentar
segurançaalimentar
higiene-segurança
higienasegurança
controlo-qualidade
controloqualidade
...
$ curl https://raw.githubusercontent.com/trickest/resolvers/refs/heads/main/resolvers-trusted.t
  ↳ xt
  ↳ -0
$ cat extracted.txt llm_subs.txt | sort -u > wordlist.txt
$ puredns bruteforce wordlist.txt example.com --resolvers resolvers-trusted.txt

```

Listing 9: Using PureDNS to resolve subdomains with the combined wordlist

Another interesting tool is [ScrapeGraphAI](#), which makes it easy to extract information about a target using headless browsers. Please note that it does not support Gemini free-tier API keys at this moment.

```

import sys
from scrapegraphai.graphs import SmartScraperGraph

graph_config = {
    "llm": {
        "model": "gpt-4o",
    },
    "verbose": False,
    "headless": True,
}

smart_scraper_graph = SmartScraperGraph(
    prompt="Extract useful information from the webpage, including a description of what the
           → company does and technology information.",
    source=sys.argv[1],
    config=graph_config
)

result = smart_scraper_graph.run()

import json
content = result.get("content")
if content:
    print(content)

```

Listing 10: ScrapeGraphAI script example

And you could also combine this with NLTK to generate subdomains from keywords:

```

$ pip3 install scrapegraphai
$ playwright install
$ python3 scrape.py https://ethiack.com
$ python3 nlp.py content.txt > extracted.txt
$ cat extracted.txt

vulnerability
management
...

```

Listing 11: Combining ScrapeGraphAI with NLTK

At this point, you may now proceed and try to solve the second CTF challenge on ai4eh.ethiack.ninja.

3.2 Screenshot Analysis

Some tools, such as gowitness [12] are essential for security researchers to explore any attack surface. It uses Chrome Headless to generate screenshots of web interfaces using the command line.

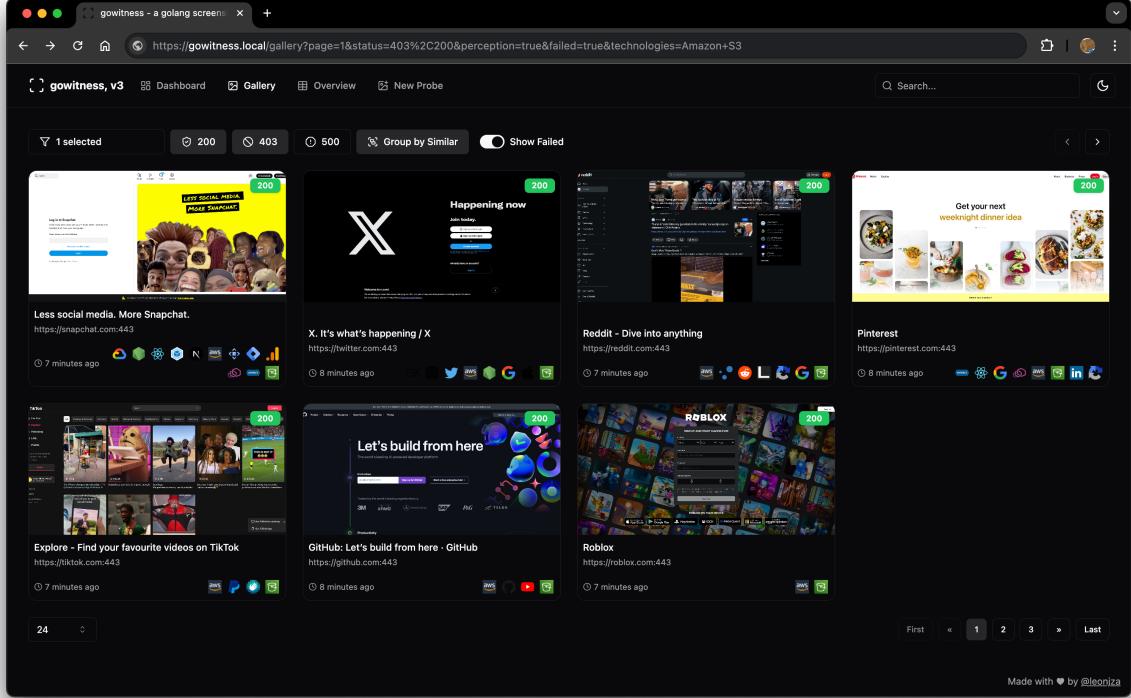


Figure 8: Gowitness Gallery

3.2.1 Eyeballer

When there are too many screenshots, it can be difficult to find sketchy applications that may be more interesting to find vulnerabilities. BishopFox has created an AI tool to solve this: eyeballer [13], a convolutional neural network for analyzing screenshots, leveraging powerful libraries like tensorflow, scikit-learn, among others.

```
$ subfinder -d anthropic.com | httpx -screenshot
$ mkdir screenshots
$ cp output/screenshot/** screenshots
$ python3 eyeballer.py --weights bishop-fox-pretrained-v3.h5 predict screenshots
...
Output written to results.csv
HTML written to results.html
```

Listing 12: Using Eyeballer to classify screenshots

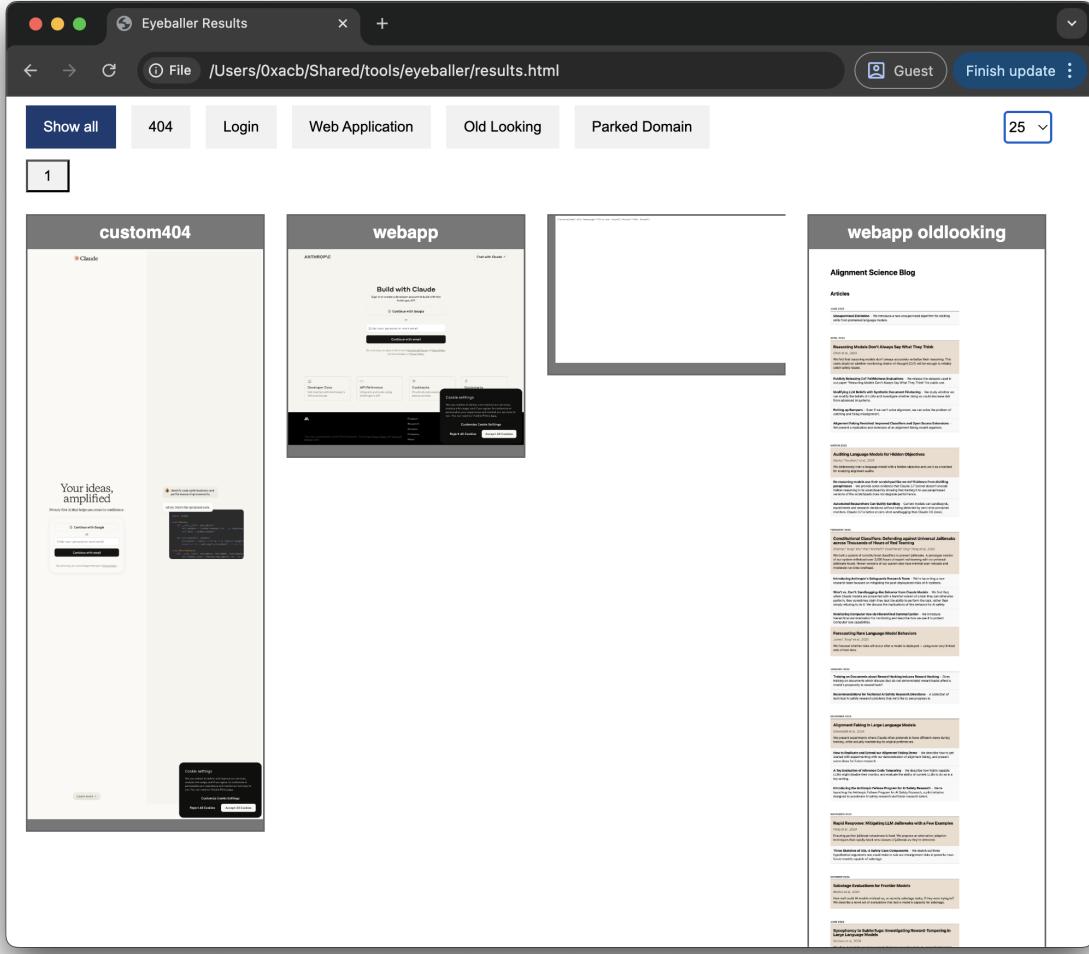


Figure 9: Eyeballer results for anthropic.com

It is also easy to train eyeballer to classify screenshots into custom categories, but you will need a dataset with a recommended size of at least 10,000 screenshots and labels. Check the repository for more information.

3.2.2 Multimodal LLMs

Most APIs have multimodal LLMs available with vision capabilities. As an alternative to eyeballer, we can also leverage them to classify screenshots, obviously for a higher price when running this at scale. However, there are local alternatives you can experiment with via Ollama, e.g. [LLaVa](#), the Large Language and Vision Assistant. Here is an example with Gemini:

```
import os
import json
from pathlib import Path
import google.generativeai as genai
from PIL import Image
from collections import Counter
import time

class ScreenshotClassifier:
    def __init__(self, api_key, screenshots_folder="screenshots"):
        genai.configure(api_key=api_key)
```

```

self.model = genai.GenerativeModel('gemini-2.0-flash')
self.folder = Path(screenshots_folder)
self.categories = ["Social Media", "Code/Development", "Documentation/Text",
                   "Gaming", "Video/Media", "Settings/System", "Email/Communication",
                   "E-commerce/Shopping", "Error/Alert", "Other"]
self.results = {}

def classify_image(self, image_path):
    try:
        image = Image.open(image_path)
        prompt = f"Classify this screenshot into one category: {', '.join(self.categories)}.\n"
        prompt += "→ Respond with only the category name."
        response = self.model.generate_content([prompt, image])
        category = response.text.strip()

        # Fuzzy match if exact match fails
        if category not in self.categories:
            for cat in self.categories:
                if cat.lower() in category.lower() or category.lower() in cat.lower():
                    return cat
            return "Other"
        return category
    except Exception as e:
        print(f"Error processing {image_path.name}: {e}")
        return "Error"

def run(self, organize_files=False, save_json=True):
    """Main method to classify all images and optionally organize them"""
    if not self.folder.exists():
        print(f"Folder '{self.folder}' not found!")
        return

    # Get all image files
    image_files = [f for f in self.folder.iterdir()
                  if f.suffix.lower() in ['.png', '.jpg', '.jpeg', '.gif', '.bmp', '.webp']]

    if not image_files:
        print("No images found!")
        return

    print(f"Classifying {len(image_files)} images...")

    # Classify images
    for i, img_path in enumerate(image_files, 1):
        print(f"[{i}/{len(image_files)}] {img_path.name}")
        category = self.classify_image(img_path)
        self.results[img_path.name] = category
        time.sleep(0.5) # Rate limiting

    # Print summary
    counts = Counter(self.results.values())
    print(f"\nSummary ({len(image_files)} total):")
    for category, count in sorted(counts.items()):
        print(f"  {category}: {count}")

    # Organize files if requested
    if organize_files:
        self._organize_files()

    # Save results
    if save_json:
        with open("classification_results.json", 'w') as f:
            json.dump(self.results, f, indent=2)

```

```

        print("\nResults saved to classification_results.json")

    def _organize_files(self):
        """Move files into category folders"""
        for filename, category in self.results.items():
            if category == "Error":
                continue

            category_folder = self.folder / category
            category_folder.mkdir(exist_ok=True)

            src = self.folder / filename
            dst = category_folder / filename

            if src.exists() and not dst.exists():
                src.rename(dst)
                print(f"Moved {filename} to {category}/")

    def main():
        api_key = os.getenv("GEMINI_API_KEY")
        if not api_key:
            print("Set GEMINI_API_KEY environment variable")
            return

        classifier = ScreenshotClassifier(api_key)
        classifier.run(organize_files=False, save_json=True)

    if __name__ == "__main__":
        main()

```

Listing 13: Script example to classify screenshots with Gemini

```

$ pip3 install google-genai google-generativeai
$ export GEMINI_API_KEY=xxx
$ python3 llm_screenshot_classifier.py
Classifying 39 images...
[1/39] cb1306110266abbef14f4857e1273112ea8c365c.png
...
Summary (39 total):
  Code/Development: 3
  Documentation/Text: 4
  E-commerce/Shopping: 1
...
Results saved to classification_results.json

```

Listing 14: Running custom Gemini screenshot analysis script

3.3 Content Discovery

LLMs are also interesting for predicting potential paths or files that can exist in the context of web applications. The same applies to extensions while fuzzing files to make the process faster and more efficient.

3.3.1 FFUFAI

FFUFAI is a wrapper for the popular web fuzzer FFUF by [Joseph Thacker](#). It automatically suggests file extensions for fuzzing based on the target URL and its headers [14]. An OpenAI or

Anthropic keys are required at this moment, since it does not support Gemini or running local models.

```
$ git clone https://github.com/jthack/ffufai
$ cd ffufai
$ pip3 install requests openai anthropic
$ chmod +x ffufai.py
$ python3 ffufai.py -u https://example.com/FUZZ -w /path/to/wordlist.txt
```

Listing 15: Installing and running ffufai

```
> ./ffufai.py -u https://josephthacker.com/FUZZ -w ~/git/SecLists/Discovery/Web-Content/api/actions.txt
{'extensions': ['.html', '.htm', '.php', '.js']}

      _/\_ \_/\_ \_/\_
     / \ \ / \ \ / \ \ / \
    / \ \ / \ \ / \ \ / \
   / \ \ / \ \ / \ \ / \
  / \ \ / \ \ / \ \ / \
 / \ \ / \ \ / \ \ / \
v2.1.0-dev

:: Method      : GET
:: URL         : https://josephthacker.com/FUZZ
:: Wordlist    : FUZZ: /Users/rez0/git/SecLists/Discovery/Web-Content/api/actions.txt
:: Extensions  : .html .htm .php .js
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500
```

Figure 10: FFUFAI execution example

3.3.2 Contextual wordlists

If you explored the previous section on subdomain enumeration, you will easily realize how contextual wordlists can also be generated to fuzz paths. The idea is quite simple: grab context from the application or web server you are fuzzing and generate a custom wordlist. Then you can use FFUF, dirsearch, or other content discovery tools with such a wordlist.

```
$ curl https://app.example.com -L -o html_content.txt
$ echo "Based on the following HTML source code from a target web application, analyze the
  ↪ technology stack, business context, naming conventions, and structure to generate a
  ↪ wordlist of 500 unique directory and file entries for fuzzing that are specifically
  ↪ tailored to this target and not found in common wordlists like seclists. Return only the
  ↪ wordlist with one entry per line, including both directories without extensions and
  ↪ files with appropriate extensions based on the identified technology stack. Return only
  ↪ the wordlist and nothing else: $(cat html_content.txt)" | llm -m gemini-2.0-flash >
  ↪ wordlist.txt
...
api/v2
b2b
2020
2022
2003
2015
pme_lider.pdf
charcutaria
industrialpt
alimentos
$ ffuf -w wordlist.txt -u https://app.example.com/FUZZ
```

Listing 16: Generating fuzz wordlist from the content of the root webpage

This is just a simple dummy example. We can expand this to consume the output of other tools, such as technology detection information, and pass those as the LLM input and use structured outputs for consistency. The possibilities are endless and more expensive models often yield the best results.

At this point, you may want to try the third challenge at ai4eh.ethiack.ninja.

4 Exploit Generation

LLMs can also be useful in generating exploits or automated detection templates if there is enough context. Providing patch diffs also seems to work to make automatic exploit generation more feasible. This capability has been demonstrated with real-world vulnerabilities, where AI models can analyze code changes and security advisories to produce functional exploit code. The effectiveness of AI-generated exploits often depends on the quality and completeness of the vulnerability information provided as context.

4.1 Nuclei AI flag

Nuclei is capable of generating and running templates using ai prompts. Although the following example will lead to false positives, giving precise prompts or manually adjusting the template will improve the results.

```
# signup at https://cloud.projectdiscovery.io/sign-up to get an API key
$ export PDPCP_API_KEY=xxx
$ nuclei -ai "find template injection"

-- -- / / ( )
/ -- \ / / / _ / - \ /
/ / / / / / / _ / / _ / /
/_ / / \_,_,/_\_,/_\_,/_ / v3.4.5

projectdiscovery.io

...
http:
- raw:
  - |
    POST /example HTTP/1.1
    Host: {{Hostname}}
    Content-Type: application/x-www-form-urlencoded

    input={{payload}}


payloads:
payload:
- "{{7*7}}"
- "{{7*'7'}}"
- "${7*7}"
- "#{{7*7}}"
- "<%= 7*7 %>"


matchers-condition: or
matchers:
- type: word
  part: body
  words:
    - "49"

- type: regex
  part: body
  regex:
    - "\d{2}"
```

Listing 17: Nuclei AI flag example for template injection

4.2 One-day exploit generation

A blog post describes how AI models and tools (GPT-4 and Cursor) can be used to create a working exploit for CVE-2025-32433, an Erlang Secure Shell (SSH) vulnerability, before any public proof-of-concept code was available. The author demonstrates how AI was able to analyze the vulnerability by comparing code diffs between vulnerable and patched versions, understand the security flaw (unauthenticated message handling), and generate and debug a functional python exploit script that successfully writes a file to the target system. [15].

```

> python3 CVE-2025-32433.py
PROOF OF DEFENSE

CVE-2025-32433 - Erlang SSH Pre-Auth Command Injection

[*] Started at: 2025-04-17 18:08:57
[+] Connected to 127.0.0.1:2222
[+] Received banner: SSH-2.0-Erlang/5.1.4.7
[+] SSH_MSG_KEXINIT packet sent
[+] SSH channel open request sent
[+] Command injection payload delivered

[!] Exploitation successful!
[+] Command executed pre-authentication: file:write_file("/lab.txt", <<"pwned">>).
[+] If the server is vulnerable, file should be created at: /lab.txt

[+] Received response from server
[+] Server response:

```

Figure 11: Executing the AI-generated CVE-2025-32433 exploit

It is also interesting that hallucinated Proof of concept (PoC)s are being published on GitHub, such as [this exploit](#) for Exim that does not make any sense. While the exploit attempts to exploit by injecting a command into `MAIL FROM:<;PAYLOAD;>`, the CVE is about an out-of-bounds buffer write.

5 Hackbots

A hackbot can be defined as *any automated system that uses AI in a meaningful way in order to find vulnerabilities* [16]. Multiple organizations are researching this topic, Ethiack is one of them. Hackbots can be applied to both Capture the Flag (CTF) scenarios and real-world applications. Hackbots can automate repetitive tasks, scan significant amounts of code, and identify potential vulnerabilities at a much faster pace. However, human judgment remains crucial. Hackbots lack natural creativity and critical thinking [7].

5.1 CAI

Cybersecurity AI (CAI) is one of the first open AI systems available designed to help anyone perform offensive security testing. The researchers found that their AI can solve security challenges much faster and cheaper than human experts, sometimes more than 3,000 times faster. What makes this impressive is that CAI actually competed against real security professionals in international competitions and won first place among AI teams, even earning prizes. CAI is an open source hackbot and we can find it on GitHub: <https://github.com/aliasrobotics/cai>.

The screenshot shows the CAI Command Reference interface. It includes:

- Agent Management (/a)**: CAI>/agent list - List all available agents; CAI>/agent select [NAME] - Switch to specific agent; CAI>/agent info [NAME] - Show agent details; CAI>/parallel add [NAME] - Configure parallel agents.
- MEMORY & HISTORY**: CAI>/memory list - List saved memories; CAI>/history list - View conversation history; CAI>/compact - AI-powered conversation summary; CAI>/flush - Clear conversation history.
- ENVIRONMENT**: CAI>/workspace set [NAME] - Set workspace directory; CAI>/config - Manage environment variables; CAI>/virt run [IMAGE] - Run Docker containers.
- TOOLS & INTEGRATION**: CAI>/mcp load [TYPE] [CONFIG] - Load MCP servers; CAI>/shell [COMMAND] or \$ - Execute shell commands; CAI>/model [NAME] - Change AI model.
- QUICK SHORTCUTS**: ESC + ENTER - Multi-line input; TAB - Command completion; r/i - Command history; Ctrl+C - Interrupt/Exit.
- Quick Start Workflows**:
 - CTF Challenge**: 1. CAI>/agent select redteam_agent; 2. CAI>/workspace set ctf_name; 3. CAI> Describe the challenge...
 - Bug Bounty**: 1. CAI>/agent select bug_bounty_agent; 2. CAI>/model claudia-3-7-somnet; 3. CAI> Test https://example.com
- Alias0 - best model for cybersecurity**:
 - Security-Focused AI Framework**: For optimal cybersecurity AI performance, use alias0 - specifically designed for cybersecurity tasks with superior domain knowledge.
 - alias0 outperforms general-purpose models in:**
 - Vulnerability assessment
 - Penetration testing and bug bounty
 - Signature analysis
 - Threat detection
 - Learn more about alias0 and its privacy-first approach:** <https://news.aliasrobotics.com/alias0-a-privacy-first-cybersecurity-ai/>

Figure 12: CAI CLI Overview

5.1.1 Solving CTF challenges

Let us use CAI to approach the first CTF challenge of this workshop. Please, remember that CTFs are an excellent learning opportunity to understand offensive security and hacking concepts, so if you are in the process of learning these skills, you should consider working through challenges manually to build your foundational knowledge before incorporating automated tools. *The integration of LLMs into learning environments presents a complex duality: while they enhance accessibility and personalization of education, they may inadvertently contribute to cognitive atrophy through excessive reliance on AI-driven solutions* [17].

```
$ pip3 install cai-framework
$ export GEMINI_API_KEY=xxx PROMPT_TOOLKIT_NO_CPR=1 CAI_STREAM=false
$ cai
CAI> /agent select redteam_agent
```

```

Switched to agent: Red Team Agent (Parallel mode disabled)
Red Team Agent (Current Agent)
Tools
  generic_linux_command
  execute_code
  (2 regular tools)
Handoffs
...
CAI> /model gemini/gemini-2.5-flash
CAI> Try to find and exploit any vulnerability here:
  ↳ https://challenges.ethiack.ninja/good-vibes

...
[72] Agent: Red Team Agent [21:14:01 (gemini/gemini-2.5-flash)]
The objective has been achieved. The flag is flag{redacted}

```

Listing 18: Running CAI in CTF mode

As you can see, CAI was able to solve a simple challenge that could eventually happen in a real world scenario. AI can hack AI code, but can it find zero-day vulnerabilities? Yes. Will it *know* that it is a zero-day? Probably not.

XBOW security researchers demonstrated how their hackbot was able to find multiple CVEs. Their system was able to identify a stored Cross-Site Scripting (XSS) vulnerability in the 2FAuth Aegis migration functionality (CVE-2024-52597). The tool discovered that the application accepts Base64-encoded Scalable Vector Graphics (SVG) icons from migration files without proper validation, allowing malicious JavaScript to be stored and executed. After initial attempts to exploit QR code processing failed, the hackbot pivoted to target the migration endpoint `/api/v1/twofaccounts/migration`, successfully uploading a crafted SVG payload that triggered XSS when accessed through the stored file path [18]. A more recent blog post also presents more zero-day XSS discoveries in Palo Alto's GlobalProtect VPN [19].

5.1.2 Extending CAI with custom tools

Instead of programming an agent with tool calls, which is also a good exercise that I recommend if you are interested in exploring this from scratch, you can use CAI, which allows the user to integrate their own tools. As an example, you may want to provide the FFUF tool to fuzz files if the agent *thinks* that it could be useful in certain scenarios. As another example, we can integrate the notify tool by Project Discovery to notify us about any vulnerabilities or leads the agent may find:

```

from pathlib import Path
from cai.sdk.agents import Agent, Runner, OpenAIChatCompletionsModel
from cai.tools.reconnaissance.generic_linux_command import generic_linux_command
from cai.sdk.agents import function_tool
from openai import AsyncOpenAI
import asyncio

MODEL = "o3-mini"

@function_tool
async def notify_vulnerability(title: str, severity: str,
                                description: str, url: str) -> str:
    """
    This tool sends a notification with the given vulnerability details
    """
    import subprocess

    s = (
        f"Title: {title}\n"
        f"Severity: {severity}\n"
        f"Description: {description}\n"
        f"URL: {url}\n"
    )
    subprocess.run(["notify-send", "-t", "10000", "-u", "normal", s])

```

```

f"{{title}}\n"
f"Severity: {severity})\n"
f"URL: `{url}`\n"
f"Description: {description}\n"
)

tmp_file_path = "/tmp/notify_data.txt"
with Path(tmp_file_path).open("w") as f:
    f.write(s)

p = subprocess.run(["notify", "-bulk", "-data", tmp_file_path],
                  check=False, stderr=subprocess.DEVNULL)

Path.unlink(tmp_file_path)

agent = Agent(
    name="CTF agent",
    description="Agent focused on finding vulnerabilities in web application.",
    instructions="You are a security expert facing a security assessment.",
    tools=[
        generic_linux_command,
        notify_vulnerability,
    ],
    model=OpenAIChatCompletionsModel(
        model=MODEL,
        openai_client=AsyncOpenAI(),
    )
)

async def main():
    result = await Runner.run(agent, "Find XSS at https://challenges.ethiack.io/xss.php")
    print(result.final_output)
asyncio.run(main())

```

Listing 19: Extending CAI with tools

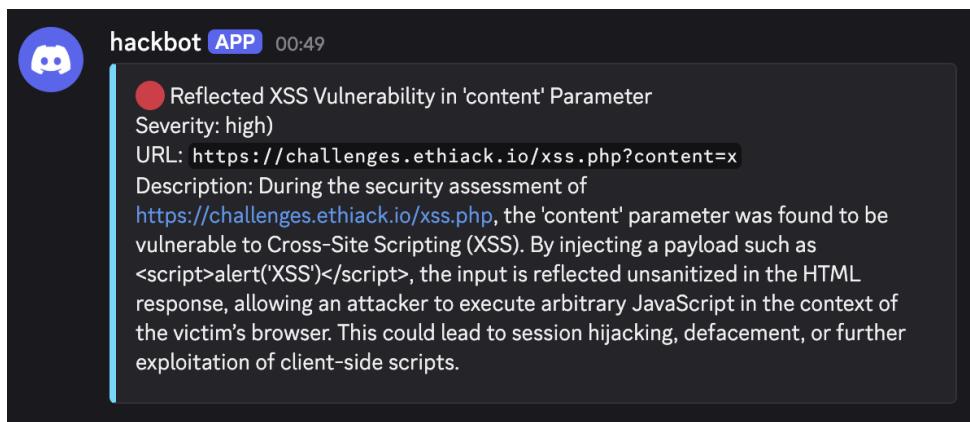


Figure 13: Resulting discord notification

5.2 Burp AI

Burp now has AI capabilities to explore potential issues that have been identified by regular scanning. These features help security researchers work faster and act as a copilot. Security professionals can choose when to use these AI features and maintain complete control over their testing work.

However, it seems that there is still room for improvement, since it does not seem able to fully exploit the PHP upload vulnerability from the first CTF challenge. First, we need to start a regular scan and then click on **Explore Issue**.

The screenshot shows the Burp Suite interface with the 'Advisory' tab selected. A single issue is listed:

- File upload functionality**

Details for this issue:

- Severity: Information
- Confidence: Certain
- URL: <https://challenges.ethiack.ninja/good-vibes>

Issue detail

The page contains a form which is used to submit a user-supplied file to the following URL:

- <https://challenges.ethiack.ninja/good-vibes>

Note that Burp has not identified any specific security vulnerabilities with this functionality, and you should manually review it to determine whether any problems exist.

Figure 14: Deterministic Issues from Crawl and Audit

The screenshot shows the AI Task Result interface. A sidebar on the left lists tasks:

- Test basic file upload functionality
- Test file download functionality
- Test path traversal in download parameter
- Test PHP file upload
- Test PHP file execution
- Test PHP file download mechanism

The main area displays the results for the first task:

Task summary

We have identified unrestricted file upload functionality that allows uploading of PHP files, however we were unable to achieve code execution as the files are stored in a non-web-accessible directory and are served only through a download mechanism that sets appropriate content-type headers and forces download rather than execution. Next steps would be to try different file extensions, attempt to bypass the download mechanism, or explore potential SQL injection vulnerabilities visible in the file list.

Test basic file upload functionality

I'm starting with a basic file upload test using a simple text file to understand:

- If the upload functionality works
- Where uploaded files are stored
- How the application responds to successful/failed uploads
- What information is revealed in the response

This is the most logical first step as it establishes baseline behavior before attempting any malicious payloads. Understanding how the application handles normal files will help inform our subsequent attack strategies.

Request	Response
<pre>1 POST /good-vibes HTTP/1.1 2 Host: challenges.ethiack.ninja 3 Content-Type: multipart/form-data; boundary=-----1234567890 6 Connection: close 7 Content-Length: 185 ...</pre>	Expand

Test file download functionality

From the previous step, we learned:

- File uploads are working successfully
- Files are stored and accessible through a download parameter
- We can see a list of previously uploaded files, including some that appear to be SQL injection attempts
- The application reveals the exact filename in both the success message and file list
- Files are downloaded via a GET parameter "download=

Figure 15: AI Task Result

#	Time	Tool	Method	Host	Path	Query	Param count	Status code	Length	Start response timer	Comment
4102	16:01:07 24 Jun 2025	Burp AI	POST	challenges.ethiack.ninja	/good-vibes		2	200	13591	61	
4103	16:01:22 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=.../etc/test.txt	1	200	352	58	
4104	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=.../etc/p...	1	200	13486	95	
4105	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=...%2F.%2C...	1	200	13486	89	
4106	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=.../.../.../...	1	200	13486	57	
4107	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=%2E%2E...	1	200	13486	92	
4108	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=.../etc/h...	1	200	13486	89	
4109	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=...%2F...	1	200	13486	88	
4110	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=.../.../proc...	1	200	13486	57	
4111	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=...%2F.%2C%	1	200	13486	57	
4112	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=.../.../var/w...	1	200	13486	92	
4113	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=...%2F...	1	200	13486	56	
4114	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=php/filter/...	1	200	13486	52	
4115	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=php/filter/...	1	200	13486	51	
4116	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=php/filter/...	1	200	13486	56	
4117	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=.../.../proc/...	1	200	13486	57	
4118	16:01:38 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=...%2F...	1	200	13486	52	
4119	16:01:50 24 Jun 2025	Burp AI	POST	challenges.ethiack.ninja	/good-vibes		2	200	13842	65	
4120	16:02:04 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes/shell.php		0	404	1339	61	
4121	16:02:15 24 Jun 2025	Burp AI	GET	challenges.ethiack.ninja	/good-vibes	download=shell.php	1	200	360	62	

Figure 16: AI Task Request History

As you can notice, the model was not clever enough to guess the uploads directory, but it still followed the intended approach and tested for path traversal.

6 MCP servers

The Model Context Protocol (MCP) is a way for AI assistants to connect to systems where data lives, including content repositories, business tools, and development environments. It was originally invented by Anthropic [20]. The main benefit is that MCP servers make it much easier to give AI tools real-world capabilities without starting from scratch each time. Instead of building complex custom connections, you can use existing servers or create new ones that work with any AI system that supports the protocol. It is like having a universal adapter that allows different AI tools to be connected to your existing systems efficiently.

MCP servers usually support capability discovery through the `tools/list` method, which allows clients to enumerate all available tools and functions that the server provides.

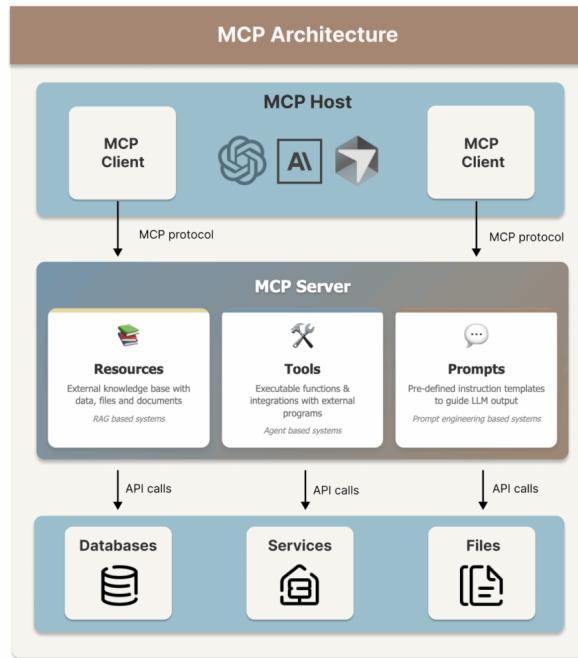


Figure 17: MCP components overview [21]

Let us take a look at some available MCP servers for offensive security. If you are curious about building your own MCP server, you may want to follow [this guide](#).

6.1 Burp Suite MCP

The Burp MCP Server is a Burp Suite extension that integrates with AI clients. Developed by PortSwigger, this extension allows security researchers to leverage AI assistants like Claude Desktop for automated actions with Human-in-the-Loop (HITL) functionality for approval or rejection.

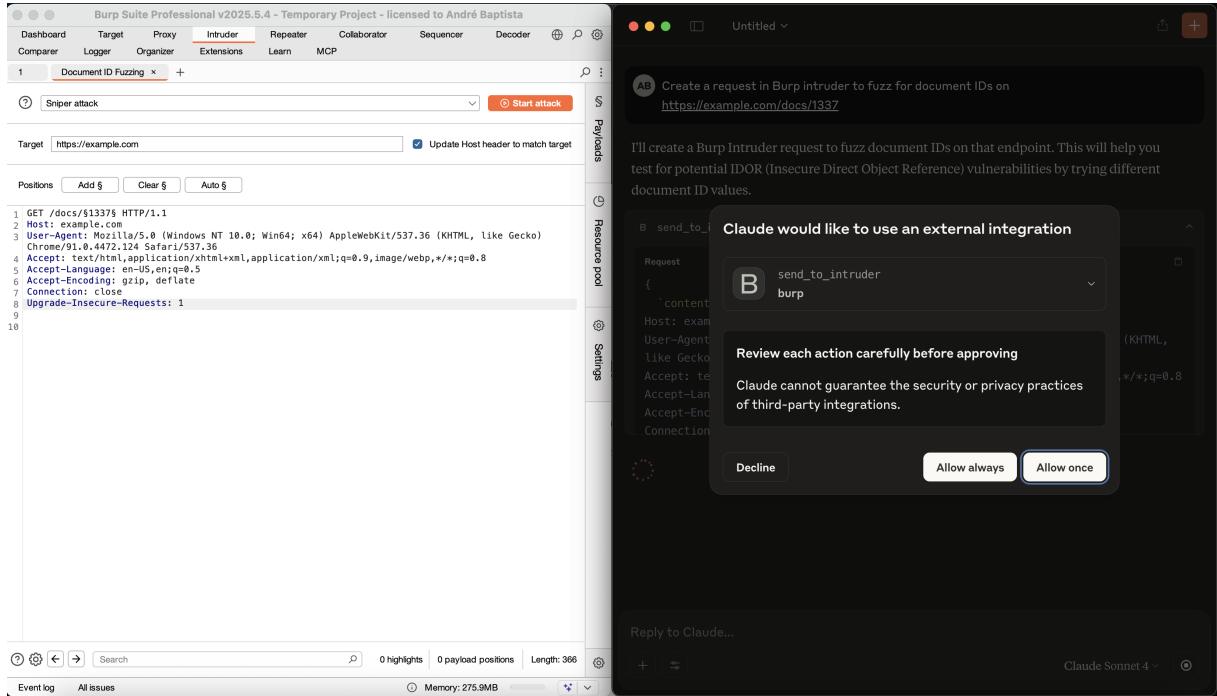


Figure 18: Burp MCP example with Claude Desktop

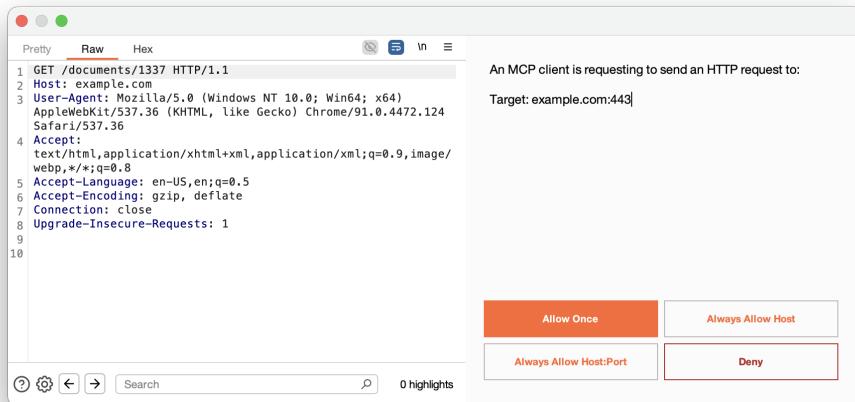


Figure 19: Burp Suite MCP approval

It is different from Burp AI as it does not directly attempt to find and exploit vulnerabilities, but rather allows users to orchestrate Burp Suite to perform actions, including, but not limited to, sending HTTP requests, creating Repeater tabs, and creating Intruder jobs. Burp Suite MCP can also be easily integrated with CAI. For more information, please refer to the CAI repository.

6.2 Other Examples

6.2.1 MCP for Security project

The [MCP for Security](#) project represents a good collection of MCP server implementations that standardize access to popular tools through a unified interface. Developed by Cyprox, it integrates multiple tools, including SQLmap for SQL injection testing, Masscan for high-

speed port scanning, FFUF for web content fuzzing, and Nuclei for template-based vulnerability scanning, among others.

6.2.2 Shift Plugin

Although [Caido](#) does not have an MCP server at this moment, there is already a really nice orchestrator by [Joseph Thacker](#) and [Justin Gardner](#): the Shift Plugin, currently available at <https://shiftplugin.com>.

Capabilities range from automatically renaming Caido Repeater tabs, sending requests, and Automate sections. At this moment, it is also capable of automatically generating and setting wordlists for fuzzing, while Burp Suite MCP seems to lack this feature.

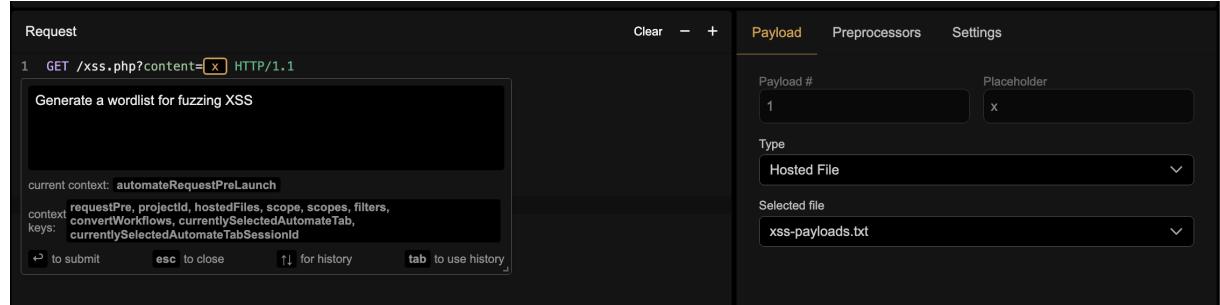


Figure 20: Shift Plugin automatically setting a fuzzing-ready wordlist

6.2.3 Ghidra MCP

The [GhidraMCP](#) project by [lauriewired](#) brings together [Ghidra](#) reverse engineering and AI models. To play with it, we just need to download the latest release, add the plugin to Ghidra, and setting up the connection between the AI client, such as Claude Desktop and Ghidra. This makes it possible for AI systems to perform reverse engineering tasks, accelerating reverse engineering tasks.

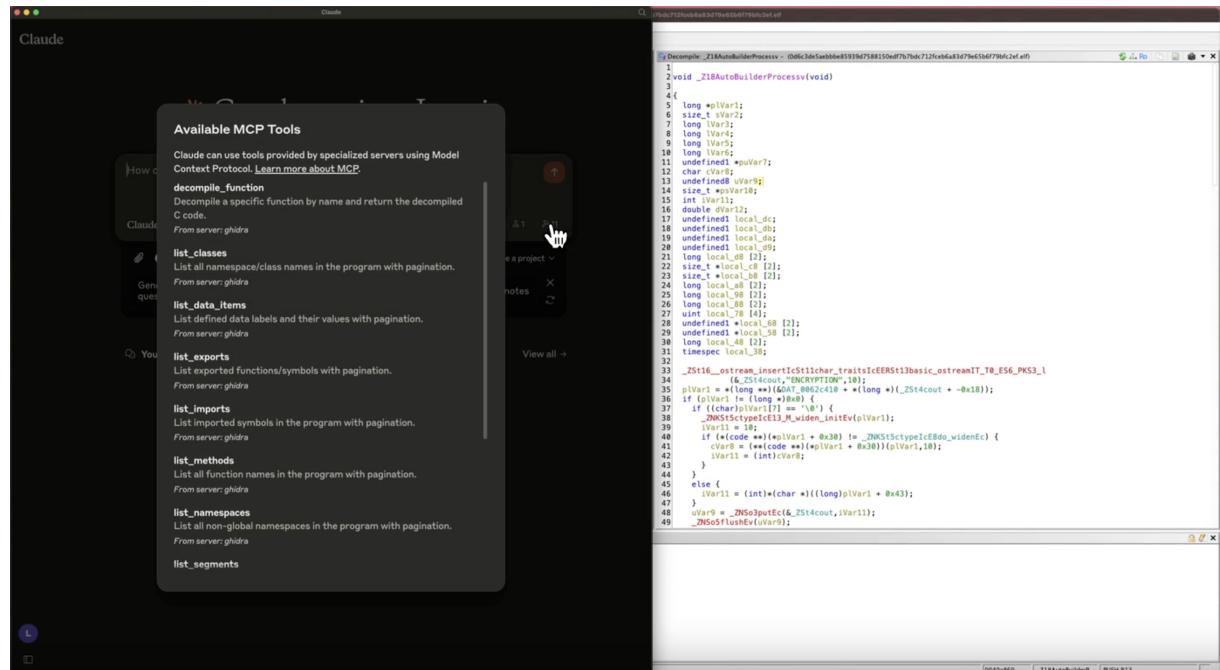


Figure 21: Ghidra MCP available functions

7 Conclusion

In this workshop, we have explored how AI is reshaping the landscape of security research, as we briefly covered how recent AI models can be leveraged to accelerate and improve the efficiency of ethical hacking. Other topics, including, but not limited to, structured outputs, fine-tuning, Retrieval-Augmented Generation (RAG), tool calling, are also interesting and accessible, so I recommend that you explore them if you want to expand your knowledge. I also challenge you to build tools that can take advantage of these topics.

As demonstrated through our practical examples, we are no longer in the realm of theoretical possibilities. These tools are here, they work, and they are being used by both security researchers and malicious actors. The question is not whether AI will change how we approach security testing, but whether we will adapt responsibly to this reality.

While these capabilities are impressive, it is important to understand that LLMs are not the holy grail solution to all our problems. We can notice impressive capabilities, but these models may hallucinate vulnerabilities that do not exist in our world, miss vulnerabilities that require deep knowledge, or struggle to build reliable PoCs. While there is room for autonomous agents, combining AI with human critical thinking in a mutualistic symbiosis is the key to effective security research.

When implementing these approaches, practical considerations also come into play. Some AI models can become expensive when used extensively. Fortunately, this barrier is fading as models become more efficient and affordable while maintaining comparable quality levels. As an alternative, you can run models locally, e.g. via Ollama, which integrates well with CAI and the LLM tool, providing an alternative way to experiment with these technologies.

Ultimately, the security community has always been driven by curiosity, creativity, and a commitment to making the Internet safer. AI provides new tools to achieve these goals, but does not change our fundamental values. This may be the beginning of that future, and it is our responsibility to use it well.

Finally, I would like to thank everyone at Ethiack, with a special acknowledgment to [Bruno Mendes](#), [Henrique Branquinho](#), [Pedro Cuco](#), and [Joseph Thacker](#) for their valuable feedback and thorough review of this work.

Until next time,
0xacb

References

- [1] Google Inc. *AI-Generated Code Statistics*. Google Developer Blog, 2024.
- [2] GitHub Inc. *GitHub Copilot Productivity and Satisfaction Survey*. <https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>, 2024.
- [3] Bruce Schneier *Secrets and Lies: Digital Security in a Networked World*, 2000.
- [4] Yuval Noah Harari *Nexus: A Brief History of Information Networks from the Stone Age to AI*, 2024.
- [5] Reddit *Cluely raised \$15M to build this, I just open sourced it and made it completely free*, 2025 https://www.reddit.com/r/SideProject/comments/1lgwb8x/cluely_raised_15m_to_build_this_i_just_open/
- [6] Harley Kimball *Disclosed Hacker Directory*. <https://www.disclosedonline.com/directory>, 2024.
- [7] HackerOne Inc. *Welcome Hackbots: How AI is Shaping the Future of Vulnerability Discovery*. <https://www.hackerone.com/blog/welcome-hackbots-how-ai-shaping-future-vulnerability-discovery>, 2024.
- [8] HackerOne Inc. *2024 Hacker Survey: AI Adoption in Security Research*. HackerOne Platform Report, 2024.
- [9] Google Inc. *Google AI Studio*. <https://aistudio.google.com/>, 2024.
- [10] Simon Willison. *LLM - A CLI utility and Python library for interacting with Large Language Models*. <https://llm.datasette.io/en/stable/setup.html>, 2024.
- [11] Hadrian Security B.V. *Subwiz: AI-Powered Subdomain Generation Tool*. <https://huggingface.co/HadrianSecurity/subwiz>, 2024.
- [12] SensePost, Orange Cyberdefense *Gowitness: A golang, web screenshot utility using Chrome Headless*. <https://github.com/sensepost/gowitness>, 2025.
- [13] Bishop Fox Inc. *Eyeballer: Convolutional neural network for analyzing pentest screenshots*. <https://github.com/sensepost/gowitness>, 2021.
- [14] Joseph Thacker *ffufai: AI-powered ffuf wrapper*. <https://github.com/jthack/ffufai>, 2024.
- [15] Platform Security *How I Used AI to Create a Working Exploit for CVE-2025-32433 Before Public PoCs Existed* <https://platformsecurity.com/blog/CVE-2025-32433-poc>, 2025
- [16] Joseph Thacker *All About Hackbots: AI Agents That Hack* <https://josephthacker.com/ai/2024/02/21/hackbots.html>, 2024
- [17] Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, Pattie Maes. *Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task*. arXiv preprint arXiv:2506.08872, 2025. <https://arxiv.org/pdf/2506.08872v1>
- [18] Diego Jurado *Stored Cross-Site Scripting (XSS) in 2FAuth*, 2024 <https://xbow.com/blog/xbow-2fauth-xss/>

- [19] Alvaro Muñoz *Breaking the Shield: How XBOW Discovered Multiple XSS Vulnerabilities in Palo Alto's GlobalProtect VPN*, 2025 <https://xbow.com/blog/xbow-globalprotect-xss/>
- [20] Anthropic *Introducing the Model Context Protocol*, 2024 <https://www.anthropic.com/news/model-context-protocol>
- [21] Towards Data Science *Model Context Protocol (MCP) Tutorial: Build Your First MCP Server in 6 Steps*, 2025 <https://towardsdatascience.com/model-context-protocol-mcp-tutorial-build-your-first-mcp-server-in-6-steps/>