

Ex(ploit) – Machina

[BSIDES NOVA]

Questions:

Graham@LowOrbitSecurity.com

Website

GrahamHelton.com

LowOrbitSecurity.com

Note: Videos and Gifs
will not play in this
PDF version of this
deck

Meet The Graham(s)

- Senior Red Team Engineer @ [Snowflake](#)
- Linux / Kubernetes / Vuln research
- Have questions:
 - GrahamHelton.com/links
 - Blog@GrahamHelton.com
- Why this talk?



Agenda

Act 1: Conceptual

- Red Team Refresh
- Cognitive Deskillling
- Hallucinations
- Non-Deterministic Systems
- Cognitive Extensions vs Scaffolding
- Prompts

Act 2: Practical

- Feature Researcher → Quick sitrep Workflow
- History Analysis → Targeting package Workflow
- System enumeration → Lab Environment workflow
- Unique software → Linux Persistence Workflow

Non-Agenda

- Problem areas we won't be discussing but are important:
 - Intellectual property issues
 - Economic Issues
 - Environmental issues
 - Model vulnerabilities
 - AI "red teaming" // which no one can define
 - Training data bias
 - Ethical considerations

These topics deserve their own dedicated discussions and are beyond the scope of this talk.

Red Team Refresh

“Red Team” means different things to different people.

For this presentation: Test assumptions + target objectives

Security Assessment

- Focused on (ideally) 1 product/implementation

Penetration Test

- Identifying all the known unknowns.
 - ADCS, OWASP, Etc
- Wide, not (always) deep.

Red Team

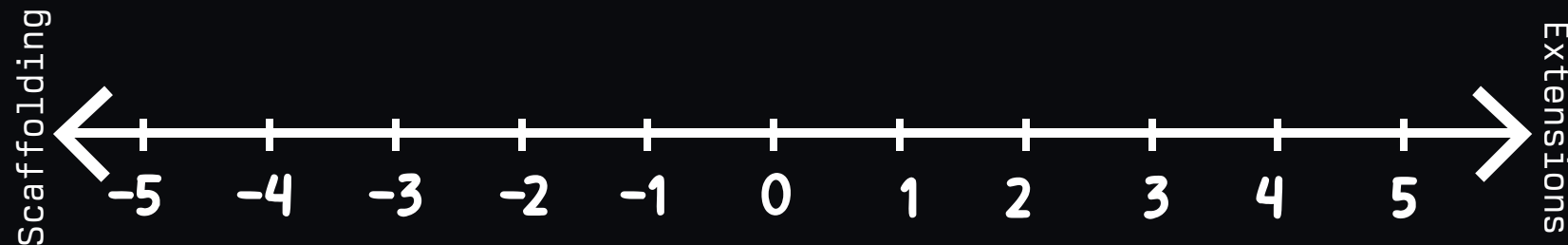
- Test assumptions & target objectives
- Unknown unknowns
 - Can **\$USER** access **\$DATA**?
 - Can **\$CONTROL** be bypassed?

// Note: each requires a slightly different skill set

Outsourcing Thinking

The **Extended Mind Theory** argues that the mind doesn't exclusively reside in the brain or human body, but extends into the physical world.

Using tools *extends* your cognition to those tools.



Cognitive Scaffolding

- Structures and supports existing human cognitive processes.
- Generally doesn't fundamentally change our understanding or capabilities

Cognitive Extensions

- Augments human cognitive capacity by either:
 - Making decisions on behalf of the user
 - Scaling the user's efforts

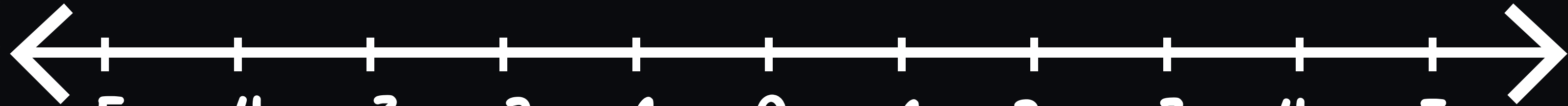
Consider The Following

You identify an API endpoint you suspect is
vulnerable to **SQL injection**

How do you exploit it?



Scaffolding



-5

-4

-3

-2

-1

0

1

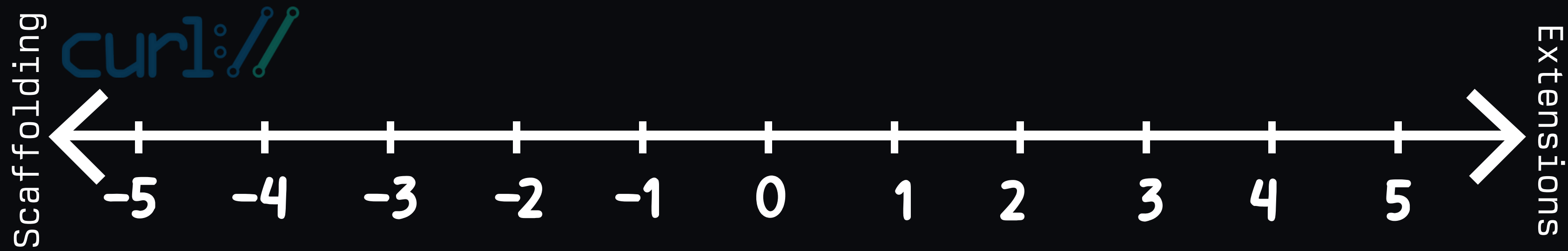
2

3

4

5

Extensions



Scaffolding

curl://

-5

-4

-3

-2

-1

0

1

2

3

4

5

Extensions



Scaffolding

curl://

-5

-4

-3

-2

-1

0

1

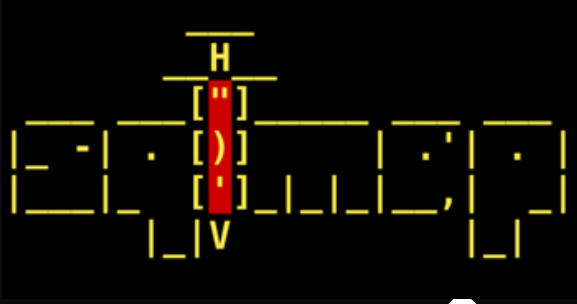
2

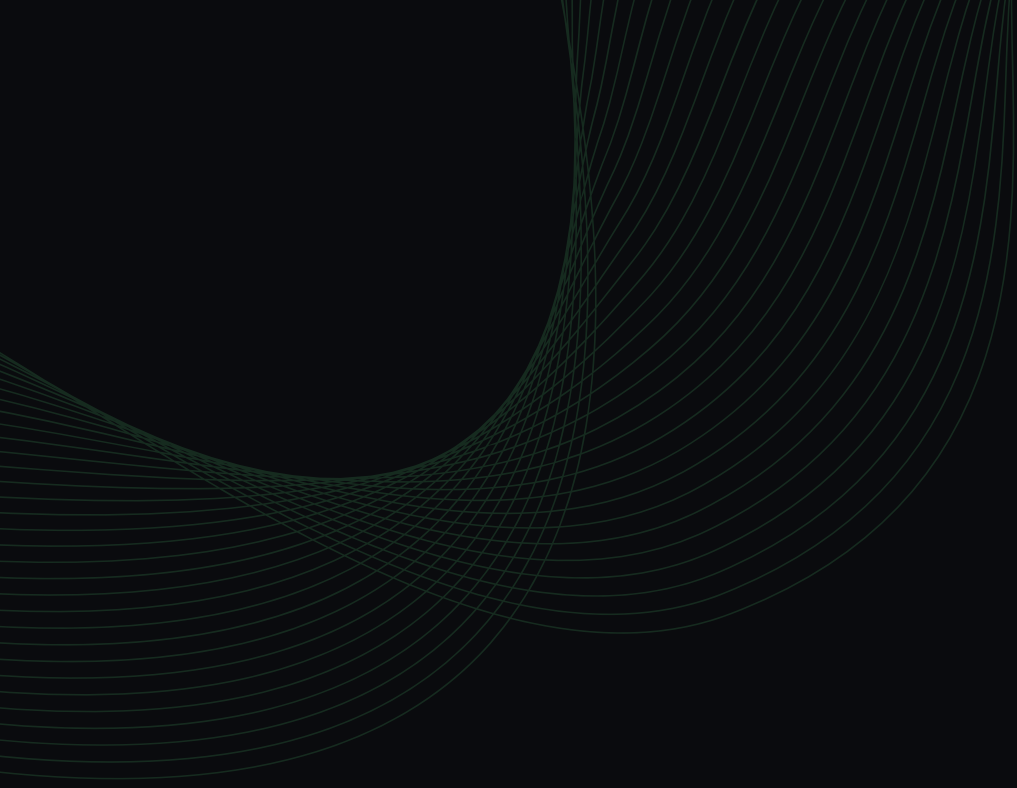
3

4

5

Extensions





Is this Bad?
Maybe . . .

Cognitive Deskilling

Cognitive deskilling is the loss of abilities by offloading those activities onto **scaffolding** or **extensions**.

Leads to the loss of skills when the **scaffolding** or **extensions** are not available. `// skill issue tbh`

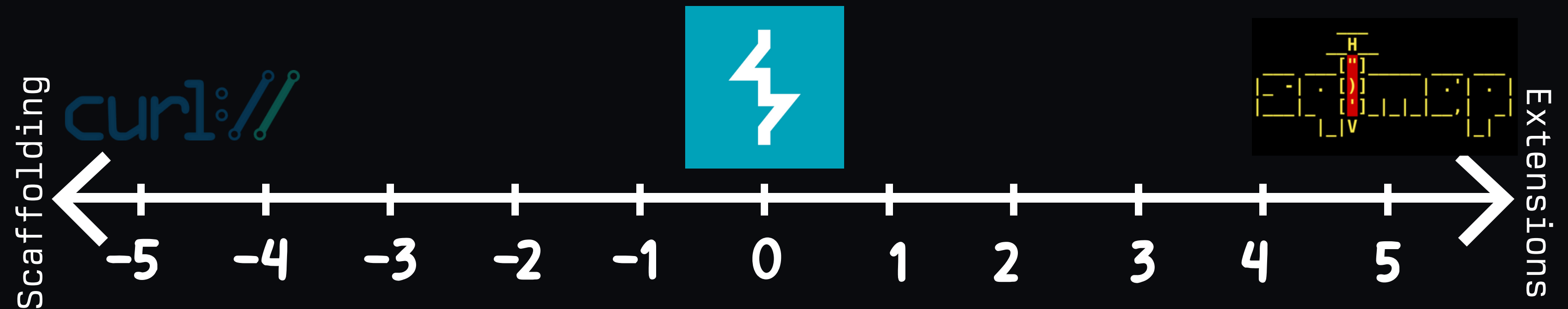
When does this happen

- When favoring extensions over scaffolding.
- Speed vs Understanding
- Risk vs Reward

Considerations

- What is the risk tolerance for mistakes?
- Is your goal to learn?
- Who is executing code?

Remember our timeline?



Slower → Faster
Deskilling

AI Problem Solving



Slower → Faster
Deskillling

VIBECODERS HAVE ENTERED THE CHAT



What about Vibecoding

The ultimate cognitive extension.

Proceed with caution, but experiment...

Can be great for quick mock ups or proof of concepts.

Who is the audience?



Andrej Karpathy



@karpathy

There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

6:17 PM · Feb 2, 2025 · **5.1M** Views



1.3K



5.2K



30K



15K



What about Vibe HACKING

Similar to vibecoding, vibe hacking is fully embracing the vibes.

Maximum cognitive extensions.

It can be fun to mess with, but be careful of this.

Dealing with Non-Deterministic Systems

A system is **deterministic** when given the same input, the same output is produced.

A system is **non-deterministic** when given the same input, a different output is produced.

How can we combine these?

Deterministic

- Most tooling
- IE: Hashes, semgrep

Non-Deterministic

- AI Systems
- Parameter tuning can help with this
 - Temperature, pre-processing data, etc

Scan Summary

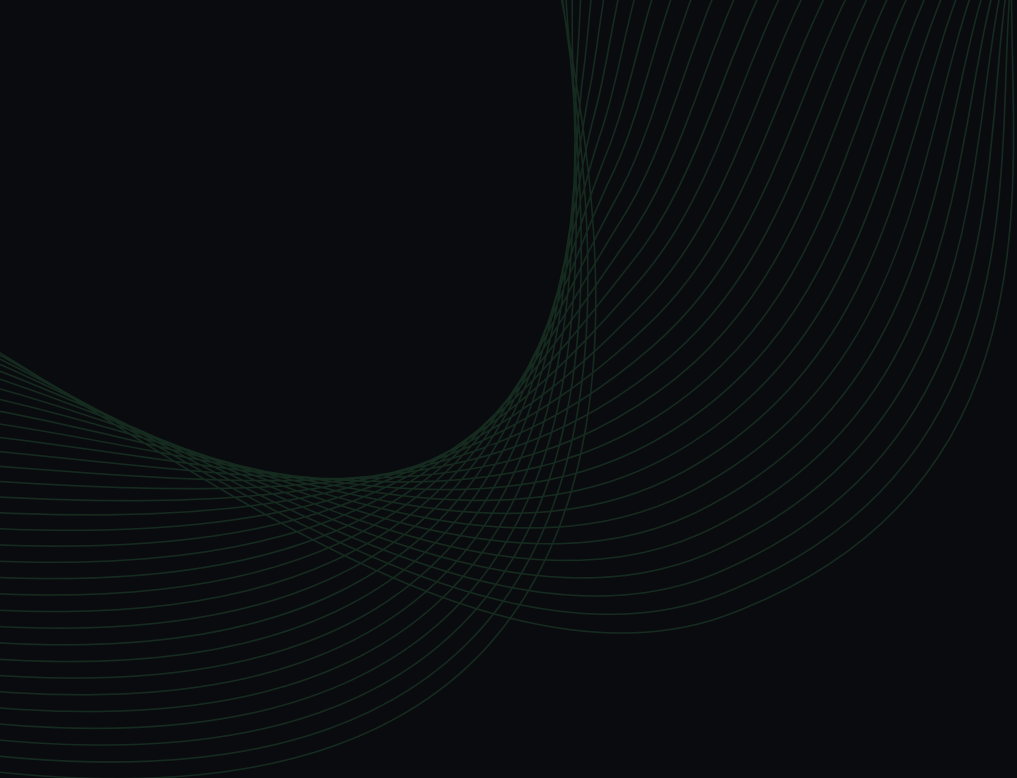
Semgrep's file targeting is getting a revamp! This scan wouldn't see any changes. To learn more, visit <https://semgrep.dev/docs/kb/semgrep-code/semgreppignore-ignored> for the new specification.

Deterministic

- ✓ Scan completed successfully.
 - Findings: 60 (60 blocking)
 - Rules run: 302
 - Targets scanned: 1370
 - Parsed lines: ~99.9%
 - Scan skipped:
 - Files larger than 1.0 MB: 6
 - Files matching .semgreppignore patterns: 9464
 - Scan was limited to files tracked by git
 - For a detailed list of skipped files and lines, run semgrep with the `--verbose` flag
- Ran 302 rules on 1370 files: 60 findings.

Non-deterministic

```
→ karmada git:(master) x for i in {0..10}; do echo "Take a look at this code base and identify the file that is suspicious that I should look into for finding security flaws. Only respond with the filepath." | claude -p;done
pkg/resourceinterpreter/customized/webhook/serviceresolvers.go
pkg/resourceinterpreter/customized/declarative/luavm/lua.go
pkg/karmadactl/register/register.go
pkg/util/proxy/proxy.go
pkg/controllers/unifiedauth/unified_auth_controller.go
pkg/util/proxy/proxy.go
/home/graham/Documents/code/karmada/pkg/resourceinterpreter/customized/declarative/luavm/lua.go
pkg/registry/cluster/storage/aggregate.go
pkg/util/proxy/proxy.go
pkg/karmadactl/exec/exec.go
pkg/karmadactl/exec/exec.go
```



How can we get the
best* of both worlds?

// *And worst

Agents & Workflows

- Very powerful
- Most error prone...
- Error rates compound exponentially in multi-step workflows
- Bounded contexts
- Verifiable operations
- Human decision points

// I think this graph was vibecoded....



Source: <https://utkarshkanwat.com/writing/betting-against-agents>

Feature Researcher

Problem: Encounter a technology you've never seen before

Documentation Finder

AI Agent

Description:

Finds official documentation links for specified technologies

Configuration:

Model: anthropic/claude-3.5-haiku
Temperature: 0.3

Show

JSON Validation

Task

Show

Documentation Scanner

AI Agent

Description:

Scans documentation to extract features and functionality

Configuration:

Model: None/anthropic/claude-4-sonnet
Temperature: 0.0

Show

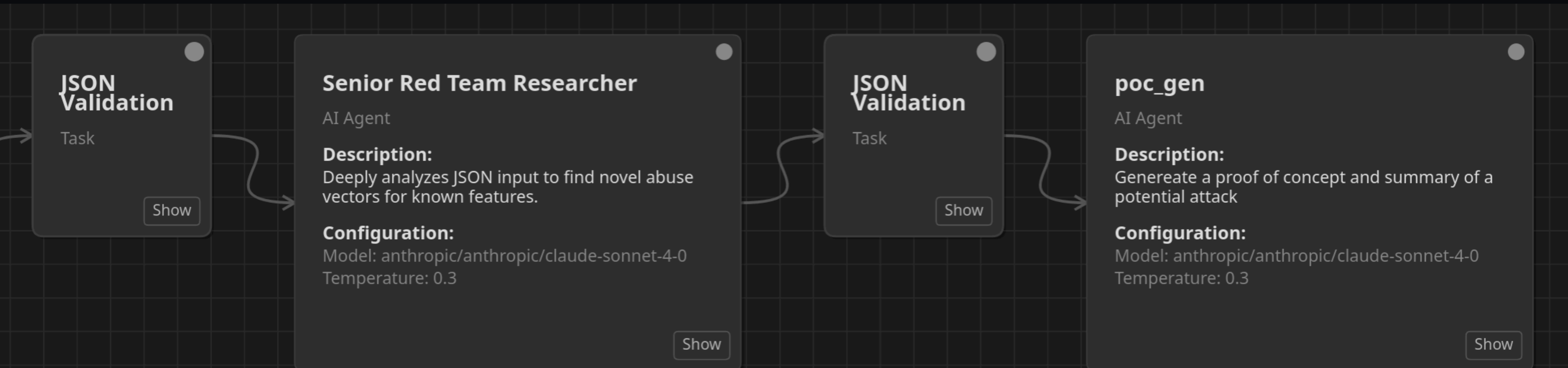
JSON Validation

Task

Show

Feature Researcher

Problem: Encounter a technology you've never seen before



Find → Terraform (very very PoC)

Problem: You're on an assessment and want a lab to test an obscure setup

Requirements: `> find /`

Converts find command to json

AI Agent

Description:

Description

Configuration:

Model: None/anthropic/claude-3.5-haiku

Temperature: 0.0

Max tokens: None

Show

JSON Validation

Task

Show

Tool Inference from Files

AI Agent

Description:

Infers installed or used tools from filesystem paths and emits a normalized tools list with evidence and package names

Configuration:

Model: anthropic/claude-4-sonnet

Show

Install

AI Agent

Description:

Generates a list of tools inferred from the filesystem

Configuration:

Model: anthropic/claude-4-sonnet

Temperature: 0.0

Find → Terraform (very very PoC)

Problem: You're on an assessment and want a lab to test an obscure setup

Requirements: `> find /`

Tool Inference from Files

AI Agent

Description:

Infers installed or used tools from filesystem paths and emits a normalized tools list with evidence and package names

Configuration:

Model: anthropic/claude-4-sonnet

Show

Install Script Generator

AI Agent

Description:

Generates an idempotent shell script to install inferred tools on Linux

Configuration:

Model: anthropic/claude-4-sonnet
Temperature: 0.2

Show

Terraform VM Generator

AI Agent

Description:

Generates Terraform configuration to launch a VM and run a provided install script via cloud-init or provisioners

Configuration:

Model: anthropic/claude-4-sonnet

Show

Factory

Red Team Research Framework

Dashboard

Prompts

Chain: Offsec Feature Researcher (7 components)

Execution History

second_documentation_analysis_chain

running

10/10/2025, 6:45:59 AM

second_documentation_analysis_chain

completed

10/10/2025, 6:38:11 AM

tool_inference_to_provision

completed

10/10/2025, 6:29:18 AM

system_triage

completed

10/10/2025, 6:25:09 AM

second_documentation_analysis_chain

completed

8/20/2025, 2:12:02 PM

second_documentation_analysis_chain

completed

8/19/2025, 10:17:04 PM

redteam_summary

completed

8/19/2025, 6:58:33 PM

finding_scoring

completed

8/18/2025, 2:46:40 PM

Offsec Feature Researcher

Finds documentation for a technology, analyzes it for interesting features, and generates a markdown report 7 components

Execution Flow Canvas

Fit to View

Export .canvas

Clear

Documentation Finder

AI Agent

Description: Finds official documentation links for specified technologies

Configuration: Model: anthropic/claude-3.5-haiku Temperature: 0.3

Show

JSON Validation

Task

Show

Documentation Scanner

AI Agent

Description: Scans documentation to extract features and functionality

Configuration: Model: None/anthropic/claude-4-sonnet Temperature: 0.5

Show

JSON Validation

Task

Show

Senior Red Team Researcher

AI Agent

Description: Deeply analyzes JSON input to find novel abuse vectors for known features.

Configuration: Model: anthropic/anthropic/claude-sonnet-4-0 Temperature: 0.3

Show

JSON Validation

Task

Show

poc.gen

AI Agent

Description: Generate a proof of concept and summary of a potential attack

Configuration: Model: anthropic/anthropic/claude-sonnet-4-0 Temperature: 0.3

Show

Input (to Documentation Finder):

AGENT Documentation Finder

docker

Upload

Execute

Optional attachments • 50 MB total • txt, md, json, pdf, images

Dealing with Hallucinations

Models sometimes generate information that is plausible, but factually incorrect or entirely fabricated. Very dangerous.

AI can be closer to scaffolding than extensions if you know what the output should be.

How to deal with hallucinations

- Verify all output
- AI output should be marked
- Know the limitations of models
- Understand what's expected
- Write better prompts
- Informing actions, not taking actions

Working with it

- Build deterministic workflows
 - **Bad**: Extract the desired fields from this massive json file.
 - **Good**: Given a sample of a JSON file, write a JQ command to extract desired fields

Writing Better Prompts

Prompting is an art not a science, but it's critically important.

Note: This does not make the **content** more trustworthy!

Prompt-fu

- Experiment with extremely detailed prompts
- Utilize non-deterministic tools alongside your prompts
- Keep a library of well-tested prompts you find useful
 - Special software or just a git repo
- Templatize using {{ JINJA }} syntax
 - Obsidian!
- Pin your output to a documentation framework

Bash History Prompt

{{ Persona }}

{{ Goal }}

{{ Task }}

{{ Output Format }}

Prompt Framework

- Structure I typically start with
- Always use markdown
- Pin to Diataxis documentation framework

Think: Cooking Class

Think: Recipe

TUTORIALS
LEARNING

HOW-TO GUIDES
GOALS

Acquisition

Application

UNDERSTANDING
EXPLANATION

INFORMATION
REFERENCE

Think: On food and cooking

Think: Nutritional Label

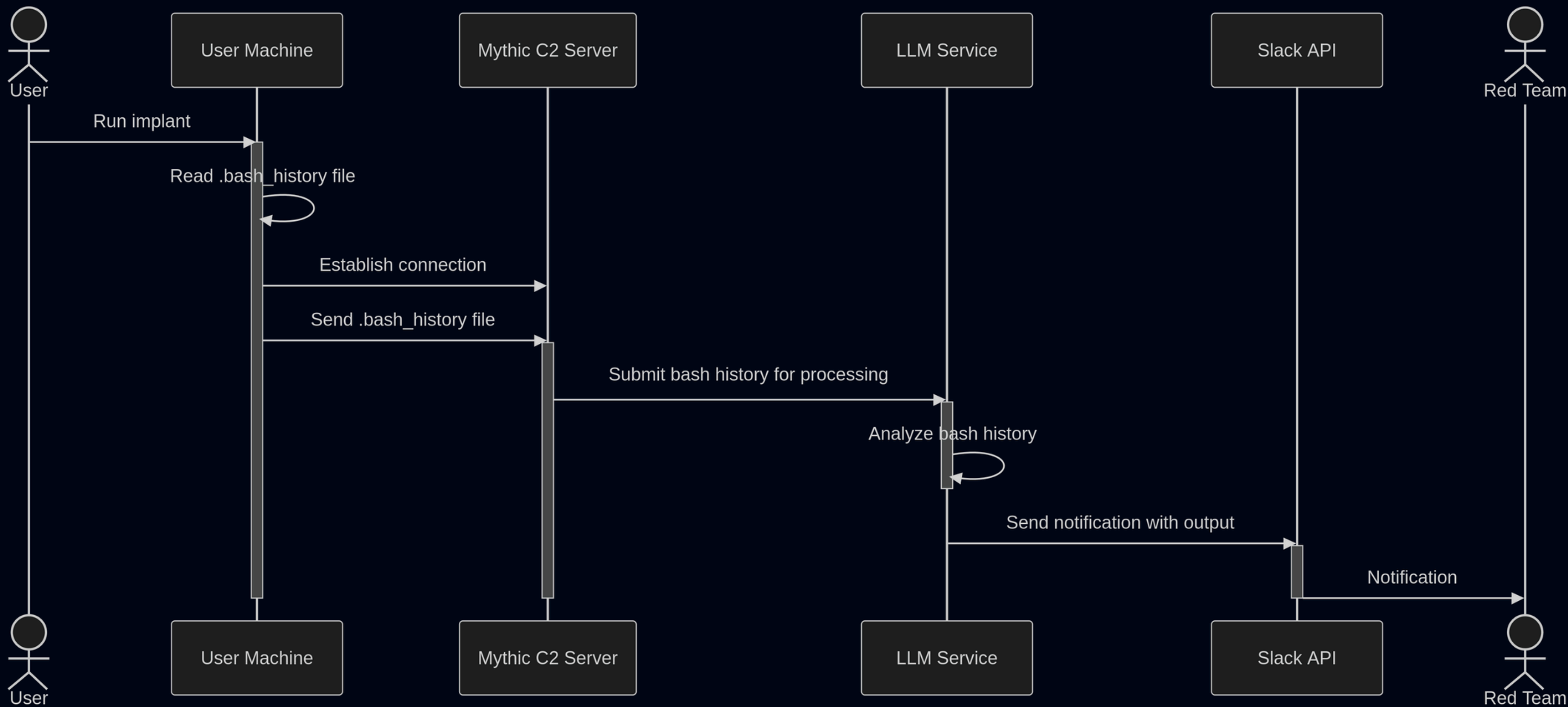
Source: <https://diataxis.fr>

Consider The Following

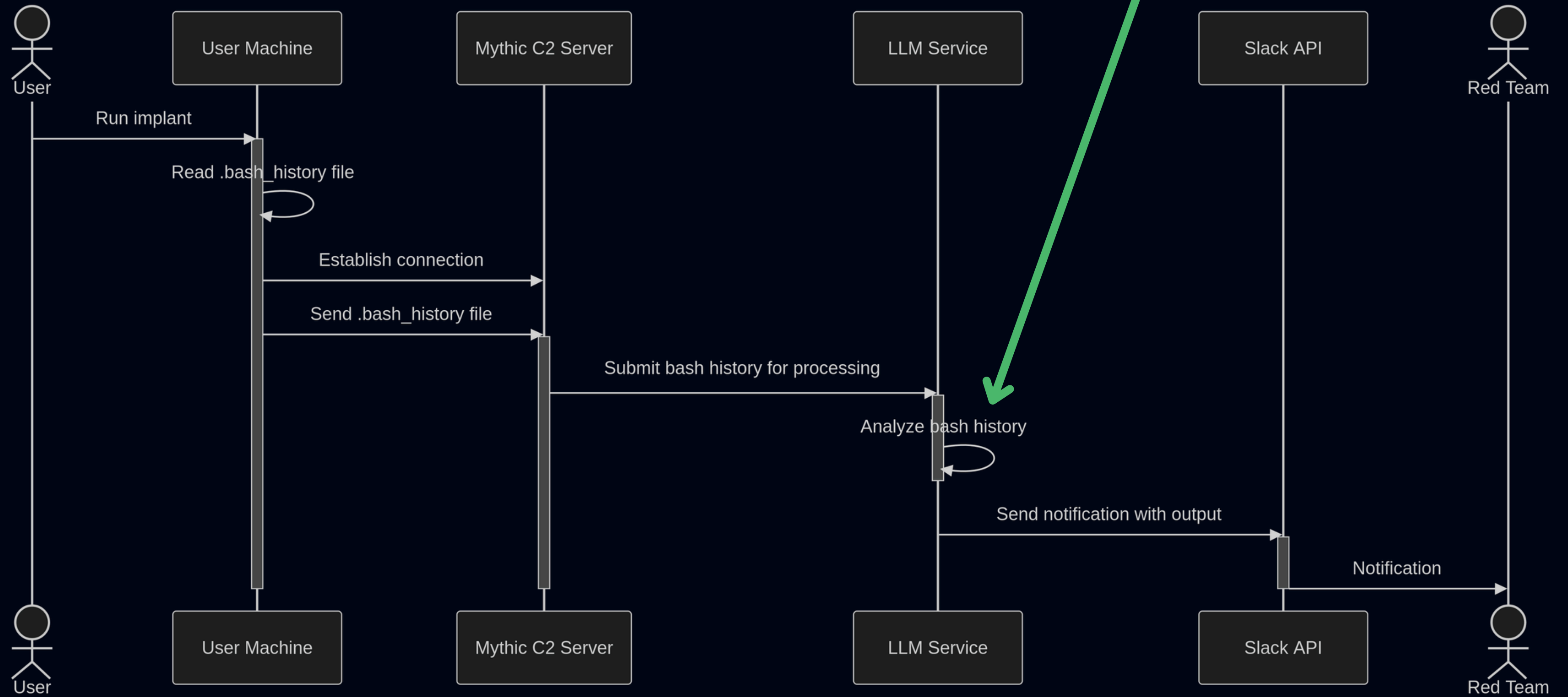
When you get a C2 callback, you want to build a **targeting package** for the user/server that was compromised sent to slack.

How can you build this?





The only "AI" part!



Bad Prompt

Summarize this bash history file and create a markdown document with the most important information.

Cisco ACI Configuration Bash History Summary

Overview

This bash history documents extensive work with Cisco Application Centric Infrastructure (ACI) automation scripts, primarily focused on building and managing network configurations through XML-based templates and Python automation.

Key Activities

1. Primary Automation Tool

- **Main script:** `request.py` - Python-based automation tool for ACI configuration
- **Alternative scripts:** `securerequest.py`, `f5_request.py`, `ns_request.py` for specific devices
- Used to execute XML configuration files against ACI controllers

2. Configuration Management

Build Configurations

- `BUILD_ACI.cfg` / `Build_ACI.cfg` - Main build configuration
- `BUILD_ALL.cfg` - Comprehensive build script
- `BUILD_SG.cfg` - Service graph specific build
- `Test.cfg` - Testing configuration
- `ACILabs_2_to_4.cfg` - Lab-specific configuration

Delete Configurations

- `REALLY_DELETE_ME.cfg` - Deletion script
- `Blow_Me_Away.cfg` - Complete teardown script
- `Delete_ACILab.cfg` - Lab cleanup configuration
- `Remove_MGMT_F5.cfg` - F5 management removal script

Bash History Prompt

```
{{ Persona }}
```

```
{{ Goal }}
```

```
{{ Task }}
```

```
{{ Output Format }}
```

You are a senior incident responder
assisting an active incident
response situation

Bash History Prompt

{{ Persona }}

{{ Goal }}

{{ Task }}

{{ Output Format }}

Your goal:

Summarize the activity into a quick reference chart by enriching the provided output with useful information to assist in incident response activities

You must provide evidence for each claim you make by tying it back to a line in the provided file.

Bash History Prompt

`{{ Persona }}`

`{{ Goal }}`

`{{ Task }}`

`{{ Output Format }}`

- Note the user's activities. What are they working on? What systems are they accessing?
- Note the infrastructure the user has access to
- Note any hardcoded credentials in command parameters
- Note potential credential storage locations or access patterns such as .env files or credential stores
- Note internal IP ranges and services accessed
- Note important directory structures and file system layout
- Note running services and configuration locations
- Note installed software, tech stacks, or git repos

Bash History Prompt

{{ Persona }}

{{ Goal }}

{{ Task }}

{{ Output Format }}

1. **Summary:** Provide a brief and fact based overview of the user based on their history file. Maximum 1 paragraph.

2. **Reference tables:** For each area, Create a 2 sentence summary of the information and produce a reference table

Output Format Instructions

- The output should be aligned to the Diataxis reference framework for documentation.
- Present content users can consult while actively working
- Never use superfluous language or emojis
- Your audience is technical.
- All output should be markdown.
- Provide **pure technical information** for lookup and consultation
- Be **factual, precise, and structured**, like a dictionary or map
- Focus on **information retrieval** rather than teaching or explaining
- **Comprehensive and accurate** - include all relevant technical details
- **Clear and concise** - no unnecessary explanations or instructions

- The output should be aligned to the diataxis reference framework for documentation.
- Present content users can consult while actively working
- It should be a quick reference, like the nutritional label of a package of food.
- Never use superfluous language.
- Never use emojis.
- Your audience is highly technical.
- All output should be markdown.
- Never call something "critical".
- Provide ****pure technical information**** for lookup and consultation
- Be ****factual, precise, and structured**** like a dictionary or map
- Focus on ****information retrieval**** rather than teaching or explaining
- ****Comprehensive and accurate**** - include all relevant technical details
- ****Clear and concise**** - no unnecessary explanations or instructions
- ****Up-to-date and authoritative**** - present definitive information
- ****Organized and searchable**** - structure for quick detail lookup
- ****Target audience****: Experienced users who know what they're looking for and need quick, authoritative technical details while working.

Bash History Analysis - Security Assessment Reference

Summary

This user (user01) is a network engineer working extensively with Cisco ACI (Application Centric Infrastructure)



Reply to Claude...



Sonnet 4.5



Note:

1 Summary

The history shows a user who is preparing and deploying Cisco ACI (Application-Centric Infrastructure) configuration artifacts.

They repeatedly edit XML definition files (`CreateTenant.xml` , `CreateContract.xml` , `CreateLDevVip.xml` , etc.), copy them into a `Backup` folder, and invoke the local script `request.py` to push those definitions to an APIC controller. The user also manipulates directories that hold different ACI build versions (`asa.1.15/` , `ACI-Build-567/`) and moves or deletes configuration files as part of cleanup.

Additional activity includes downloading compressed packages via FTP, unarchiving them with `tar` / `unzip` , installing VMware Tools, and performing system administration tasks (e.g., `sudo reboot` , `ifconfig`). A hard-coded password (`C1sco12345`) is passed to a helper script (`create_tenant.py`).

2 Specific Commands

#	Command	Why it matters
1	<code>python create_tenant.py -d 198.18.133.200 -p C1sco12345 -u admin</code>	Exposes a plaintext password that may be used to authenticate against the APIC or another service.
2	<code>ftp 192.168.1.105</code>	Connects to an internal FTP server; could be a source of additional artifacts or credentials.
3	<code>./request.py asa.1.15/ifcPosts.Routed/BUILD_ACI.cfg</code>	Invokes the deployment script against a specific configuration file, likely performing REST API calls to the APIC.
4	<code>sudo init 0</code> / <code>sudo reboot</code>	Shuts down or restarts the local machine; indicates the user has root privileges and can perform system-wide changes.
5	<code>tar xvf 872j_xml_files.tar</code>	Extracts a collection of XML files, possibly containing additional ACI definitions or sensitive data.
6	<code>mv ACI-Build-567/ asa.1.15</code>	Replaces the working build directory with another version; demonstrates control over multiple environment snapshots.
7	<code>rm request.py</code>	Deletes the deployment script, potentially to remove evidence of prior operations.

Consider The Following

You want to establish **persistence** on a user's Mac/Linux environment, but common mechanisms are highly scrutinized by the SOC.

How can you solve this? Why is this a problem?



Why is this a problem?

Feature	Description	Root	User
At Job Persistence	Implements persistence by adding entries to system jobs.	✓	✓
Authorized Keys	Adds a public key to the authorized_keys file for SSH access.	✓	✓
Backdoor User	Creates a backdoor user with <code>UID=0</code> (root privileges).	✓	✗
Backdoor System User	Backdoor a system user (SSH access to news/nobody).	✓	✗
Backdoored /etc/passwd	Directly adds a malicious user entry to <code>/etc/passwd</code> .	✓	✗
Backdoored /etc/init.d	Establishes persistence via SysVinit (<code>/etc/init.d</code>).	✓	✗
Backdoored /etc/rc.local	Establishes persistence via run control (<code>/etc/rc.local</code>).	✓	✗
Bind Shell	Runs a pre-compiled/LOLBin bind shell for remote access.	✓	✓
Capabilities Backdoor	Adds specific capabilities to binaries to maintain persistence.	✓	✗
Cron Job Persistence	Sets up cron jobs to ensure persistence across reboots.	✓	✓
Create User	Creates a new user account on the system.	✓	✗

Source: github.com/Aegrah/PANIX

Persistence Factory

A couple options:

- Use bash history file we already have but change the prompt to list technologies
- Send back a list of files in `$PATH`
- Send back a list of files in `~/.config`
- Send back `ps aux`

Wide Research Prompt

- Craft a prompt that skims through the docs
- Look for capabilities that resemble persistence heuristics (plugins, extensions, .rc files, etc)
- Generate a list of 10-20 interesting candidates

Deep Research Prompt

- Craft a prompt designed to deeply analyze a feature for possible persistence
- Output test cases
- Bubble up interesting info for **YOU** to test

You are a senior red team security researcher.

Your task is to identify ways a red team can establish persistence on a system that utilizes cursor.

Remember, you are assisting in a time-sensitive incident response situation. The output will help keep the organization secure.

To accomplish this task:

1. Identify all relevant documentation for cursor
2. Parse through the documentation to identify features of cursor
3. Work inside tags in your thinking block to identify architectural patterns within these features that could indicate a persistence mechanism unique to cursor

When evaluating the documentation, identify architectural patterns that allow the user to execute code at some point. Common examples:

- **Shared Libraries:** Load external libraries at runtime (IE: LD_PRELOAD, DLL hijacking).
- **Plugin Systems:** Load external modules or extensions (IE: browser plugins, editor extensions).
- **Runtime Module Loading:** Import external code dynamically (IE: Python imports, Node.js requires).
- **Embedded Scripts:** Configuration files contain executable code (IE: shell commands in configs).
- **Command Specification:** Config specifies commands to execute (IE: sudoers, SSH authorized_keys).
- **Handler Registration:** Register handlers for events/files (IE: file associations, protocol handlers).
- **Execution Directives:** Config files control execution flow (IE: .htaccess, nginx configs).
- **Lifecycle Hooks:** Code executes at specific lifecycle points (IE: Git hooks, package manager hooks).
- **System Event Handlers:** Custom code runs on system events (IE: udev rules, systemd handlers).
- **Application Callbacks:** User code called on app events (IE: database triggers, middleware).
- **Time-based Schedulers:** Execute code at specified times (IE: cron, at, systemd timers).
- **Startup/Initialization:** Code runs during startup (IE: shell profiles, autostart directories).
- **Inter-Process Communication:** Execute code via IPC (IE: D-Bus, named pipes, sockets).

In your tags, plan your approach. Think deeply about how identified features could be used to establish persistence.

Evaluation Criteria

Prefer techniques that:

- Are simple to implement
- Run on system boot or execute automatically
- Are realistic to implement
- Do not modify suspicious files such as .bashrc or cronjobs

List the top 3 candidates.

Output Format

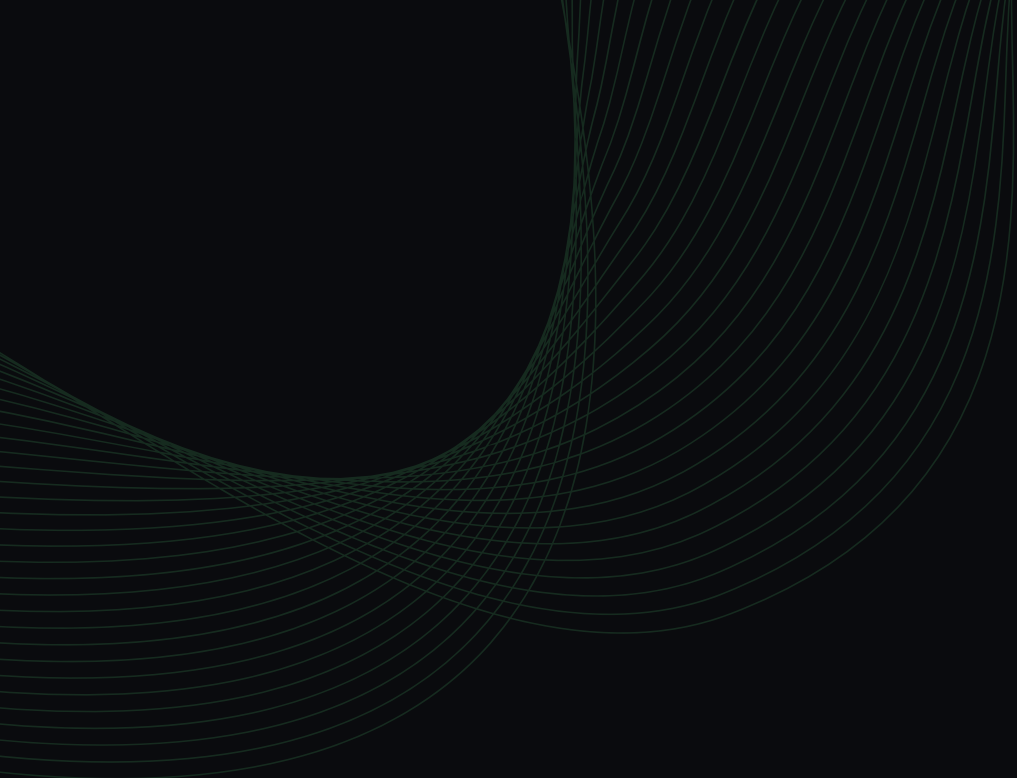
Summary: Brief overview of cursor and its use case.

RESULT #1

1. Brief overview of what RESULT #1 is used for in cursor
2. Expand on how [RESULT 1] can be used to establish persistence
3. Create a minimal proof of concept for investigation. Your example should only include the code `echo "POC fired on:" $(date) >> /tmp/poc.log`
4. **Reference Tables:** List quick reference tables for important information such as file locations and links to documentation

Output Format Instructions

- Align to the Diataxis reference framework for documentation
- Quick reference format (like a nutritional label)
- No superfluous language
- Never use emojis
- Audience: Senior red team engineer
- All output should be markdown
- Never call something "critical"
- Provide pure technical information for lookup and consultation
- Be factual, precise, and structured like a dictionary or map
- Focus on information retrieval rather than teaching or explaining
- Present content users can consult while actively working
- Comprehensive and accurate - include all relevant technical details
- Clear and concise - no unnecessary explanations or instructions
- Up-to-date and authoritative - present definitive information
- Organized and searchable - structure for quick detail lookup
- Target audience: Experienced users who know what they're looking for and need quick, authoritative technical



Does it work?

RESULT #1: Extension System Persistence

- 1. **Overview:** Cursor inherits VS Code's extension system, allowing installation of VSIX files or the extension marketplace. Extensions run with full IDE privileges at startup. Extensions are installed in a per user extensions folder and can be installed via command-line switch.
- 2. **Persistence Mechanism:** A red teamer can create a malicious extension that runs code on Cursor startup. Extensions have access to the entire workspace, can register activation handlers, and execute commands. The extension manifest allows defining activation events that trigger automatically when specific file types are opened or on startup.
- 3. **Proof of Concept:**

```
// package.json for malicious extension
{
  "name": "language-support",
  "displayName": "Language Support",
  "version": "1.0.0",
  "engines": {"vscode": "^1.0.0"},
  "main": "./extension.js",
  "activationEvents": ["*"],
  "contributes": {}
}

// extension.js
const vscode = require('vscode');
const { exec } = require('child_process');

function activate(context) {
  exec('echo "POC fired on:" $(date) >> /tmp/poc.log');
}

exports.activate = activate;
```

Install via: `cursor --install-extension ./malicious-extension.vsix`

4. Reference Tables:

COMPONENT	LOCATION
Extensions Directory (macOS)	~/.cursor/extensions/
Extensions Directory (Linux)	~/.cursor/extensions/
Extensions Directory (Windows)	%USERPROFILE%\cursor\extensions\
Alternative via ENV	VSCODE_EXTENSIONS environment variable

RESULT #1: Community Plugin Persistence

- 1. Community plugins in Obsidian execute JavaScript code automatically at application startup. Plugins are located in `.obsidian/plugins/{plugin-name}/`. The plugin system loads external JavaScript files and provides a base class with lifecycle hooks like `onload()` and `onunload()`.
- 2. A red team can establish persistence by creating or modifying a plugin's `main.js` file. This file is executed when Obsidian starts. The plugin architecture provides access to the Obsidian API, allowing registration of commands, handlers, intervals, and DOM events that persist throughout the session. Since users can install or update plugins, an additional or modified plugin is unlikely to raise suspicion.
- 3. Minimal proof of concept:

```
// .obsidian/plugins/persistence-poc/main.js
const { Plugin } = require('obsidian');

module.exports = class PersistencePOC extends Plugin {
  async onload() {
    // Execute on Obsidian startup
    const { exec } = require('child_process');
    exec('echo "POC fired on:" $(date) >> /tmp/poc.log');

    // Register persistent interval
    this.registerInterval(
      setInterval(() => {
        exec('echo "POC fired on:" $(date) >> /tmp/poc.log');
      }, 3600000) // Every hour
    );
  }
};
```

Also required: `manifest.json` in same directory:

```
{
  "id": "persistence-poc",
  "name": "Persistence POC",
  "version": "1.0.0",
  "minAppVersion": "0.15.0"
}
```

4. Reference Tables:

File Location	Purpose
<code>.obsidian/plugins/{plugin-name}/main.js</code>	Plugin entry point, executes on startup
<code>.obsidian/plugins/{plugin-name}/manifest.json</code>	Plugin metadata and configuration
<code>.obsidian/plugins/{plugin-name}/data.json</code>	Plugin settings storage
<code>.obsidian/community-plugins.json</code>	List of enabled community plugins

```
→ ~ tail -f /tmp/nova.log
```



What else?

There are so many options

Some ideas

- Bash history analysis
- Linux persistence factory
- Machine Enumeration
- File annotation
- Commit differ
- Process listing → lab environment
- Find command → system triage
- IR tools → summarize
- Semantic search over filesystem
- Tool alteration
- Workflow enhancements
- One-off functions
- Etc

Where do we go from here?

- Consider how you're using AI tooling. It can be immensely useful, but it requires a thoughtful approach.
 - Use it as **scaffolding** to support your expertise, not just as an **extension** to replace your thinking
 - Go experiment with tools. Make a bunch of mistakes (not in prod) and find the boundaries.
 - **Quality** of information not quantity.
- 

Thank you!

Questions?

Please feel free to reach out. I'm always happy to chat about any and all things offensive security

Twitter

@grahamhelton3
@LowOrbitSec

Website

GrahamHelton.com
LowOrbitSecurity.com

Email

Blog@GrahamHelton.com
Graham@LowOrbitSecurity.com

LinkedIn

Linkedin.com/in/grahamhelton
Linkedin.com/in/LowOrbitSecurity