



IdentityShield
Summit '26

Deep Dive into Building Next-Gen Local AI Security Reviewers

Hardik Mehta (hardw00t)
Rajanish Pathak (h4ckologic)



miniOrange



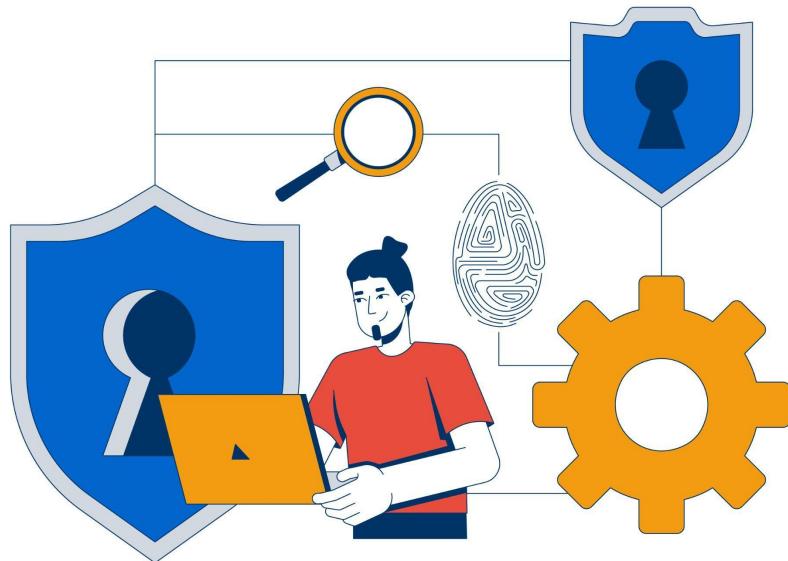
About me

I (@h4ckologic) am a security researcher passionate about uncovering and addressing critical vulnerabilities in complex technology implementations. My work includes identifying and reporting issues to top tech companies like Apple, Google, Microsoft and many others, some of my CVES identified are Apple, PhantomJS, and NPM html-pdf. I've had the privilege of sharing my research at leading conferences, including HackLu, BlackHat MEA, NoNameCon, Ekoparty, Hacktivity, Hack in the Box and Romhack. With a focus on practical solutions and deep technical insights, I'm dedicated to advancing security practices and contributing to the global infosec community.



RAJANISH PATHAK
Senior Manager
Security Services

#Product #Security #Research





Today's Journey

1. The Problem → Why traditional SAST tools falls short
2. Understanding Code → AST and semantic analysis foundations
3. Human Expert Reasoning → How security experts actually think
4. Teaching AI → Combining LLMs + RAG for code review
5. Privacy First → Why local models change everything
6. Implementation → Building a production-ready system
7. Economics → Free security reviews for life
8. Open Source → FalconEYE demonstration





THE PROBLEM

Why Traditional SAST Falls Short



The 85% False Positive Problem

Pattern Matching Is Fundamentally Limited

- Regex and signatures only find what they're programmed to find
- No understanding of context, intent, or data flow
- Cannot distinguish real vulnerabilities from safe patterns



The Result: Alert Fatigue

- 70-90% of alerts are false positives
- Developers learn to ignore security tools
- Real vulnerabilities get buried in noise



"We stopped using our SAST tool because every scan produced 500+ alerts that were 95% noise." — Senior Security Engineer



The Missing Context Problem

Traditional tools see: user_input in query

They flag: "SQL Injection!"

But they miss:

- Is user_input actually user-controlled?
- Is it sanitized upstream?
- Is the query even reachable from external input?
- What's the business context?

Without context, every potential pattern is a "vulnerability"

This is why we need semantic understanding





IdentityShield
Summit '26

UNDERSTANDING CODE

AST and Semantic Analysis

Abstract Syntax Trees: The Foundation

What is an AST?

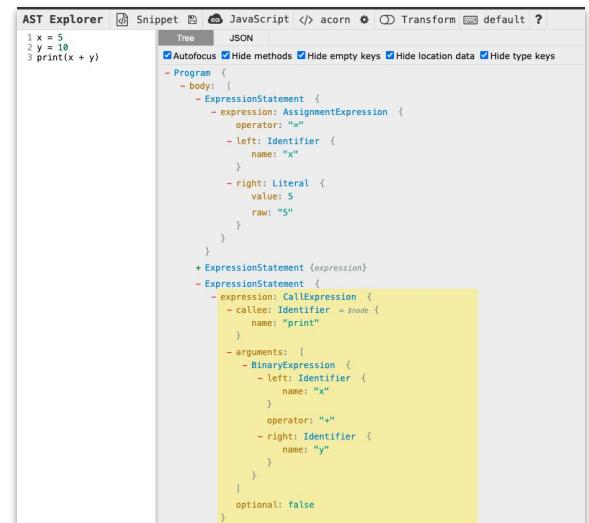
Source code transformed into a structured tree representing actual logic

Why AST Matters for Security:

- Semantic Boundaries — Know where functions, classes, scopes begin/end
- Language Agnostic — Same analysis approach across Python, JS, Go, Rust
- Data Flow Ready — Follow variables through assignments and returns
- No Regex Needed — Work with code structure, not text patterns

Tree-sitter: The Engine

- Powers VS Code, GitHub syntax highlighting
- Incremental parsing — fast on large codebases
- Supports 40+ languages with consistent API



The screenshot shows the AST Explorer interface with the following code input:

```
1 x = 5
2 y = 10
3 print(x + y)
```

The resulting JSON AST structure is:

```
{
  "Program": {
    "body": [
      {
        "ExpressionStatement": {
          "expression": {
            "AssignmentExpression": {
              "operator": "=",
              "left": {
                "Identifier": {
                  "name": "x"
                }
              },
              "right": {
                "Literal": {
                  "value": 5,
                  "raw": "5"
                }
              }
            }
          }
        }
      },
      {
        "ExpressionStatement": {
          "expression": {
            "CallExpression": {
              "callee": {
                "Identifier": {
                  "name": "print"
                }
              },
              "arguments": [
                {
                  "BinaryExpression": {
                    "left": {
                      "Identifier": {
                        "name": "x"
                      }
                    },
                    "operator": "+",
                    "right": {
                      "Identifier": {
                        "name": "y"
                      }
                    }
                  }
                ]
              ],
              "optional": false
            }
          }
        }
      }
    ]
  }
}
```



From Text to Structure

This is the foundation for semantic analysis

Source Code:

```
1 def process(user_input):
2     query = "SELECT * FROM users WHERE id=" + user_input
3     return db.execute(query)
```

AST Representation:

```
1 FunctionDef "process"
2   └─ args: ["user_input"]
3   └─ body:
4     └─ Assign "query" ← BinOp(string + user_input)
5       └─ Return: Call db.execute(query)
6
```

Now we can trace: `user_input → query → db.execute`



IdentityShield
Summit '26

HUMAN EXPERT REASONING

How Security Experts Actually Think



The Expert Security Review Process

Step 1: Context Gathering

"What does this app do? What's the threat model?"

Step 2: Data Flow Tracing

"Where does this input come from? Where does it go?"

Step 3: Pattern Recognition (Experience)

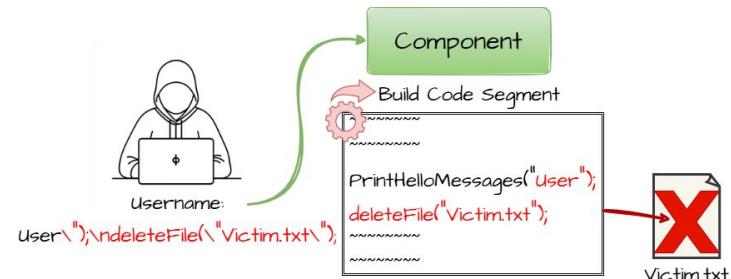
"I've seen this anti-pattern cause CVEs before"

Step 4: Risk Assessment

"What's the blast radius? Is it reachable externally?"

Key Insight: Experts don't pattern-match

They understand intent, trace flows, and reason about impact





Expert vs Traditional SAST

Traditional SAST asks:

"Does line X match regex Y?"

No context. No reasoning. Binary yes/no.

Human Expert asks:

"This user input flows through function X, gets partially sanitized in Y, but the encoding in Z doesn't handle this edge case..."

Full context. Deep reasoning. Nuanced assessment.



The Question:

Can we teach AI to think like a human security expert?

Answer: Yes. With the right architecture.



IdentityShield
Summit '26

TEACHING AI

LLMs + RAG for Security Review

Why Large Language Models?

LLMs Understand Code Semantics

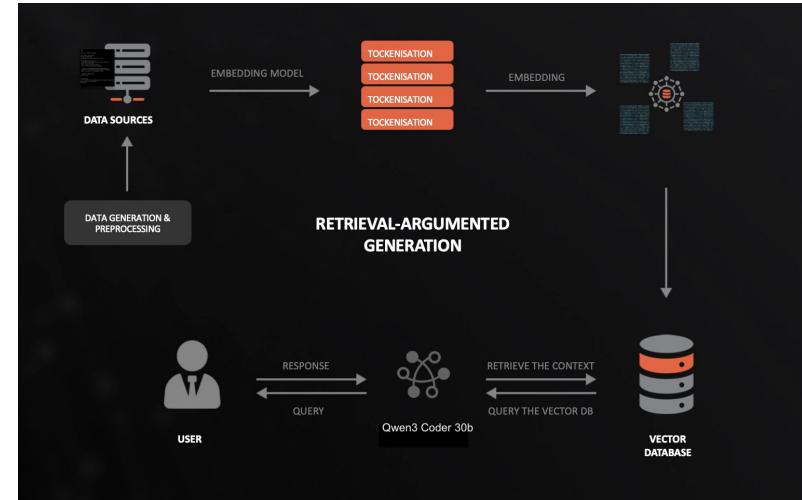
- Trained on millions of repositories
- Learned patterns, idioms, and anti-patterns
- Can reason about code intent, not just syntax

Beyond Pattern Matching:

- Understand what code is trying to do
- Recognize security-relevant patterns from training
- Generate contextual explanations
- Suggest appropriate fixes

The Gap: LLMs need context

They can't read your entire codebase in one prompt





The Complete Analysis Process

- Stage 1:
Code Ingestion**
Parse source files with Tree-sitter into AST
- Stage 2:
Intelligent Chunking**
Break code at semantic boundaries (functions, classes)
- Stage 3:
Vector Indexing**
Embed chunks and store in vector database
- Stage 4:
Context Assembly (RAG)**
For each file, retrieve architecturally relevant code
- Stage 5:
AI Analysis & Reporting**
LLM reviews with full context + validation pass,
Findings with confidence scores and fix suggestions





IdentityShield
Summit '26

PRIVACY FIRST

Why Local Models Change Everything

Why Local Large Language Models?

The Cloud Security Paradox

To analyze your code for security, you send it to someone else's servers?



Local LLMs Solve This:

- Zero data transmission — code stays on your hardware
- Complete IP protection — proprietary code remains private
- Regulatory compliance — meet data residency requirements
- Airgapped environments — works without internet

Modern Hardware Makes It Possible:

- Consumer GPUs can run capable code models
- 32GB RAM runs production-quality analysis
- Apple Silicon M-series excellent for local inference

```
l - Desktop ollama ls
NAME ID SIZE MODIFIED
qwen3-embedding:Rb 64b933495768 4.7 GB 2 days ago
qwen3-embed-text:latest 8a1097422b47 274 MB 2 days ago
qwen3-coder:30b 86c1097e7fce0 18 GB 2 days ago
qwen3-coder:latest 86c1097e7fce0 18 GB 4 weeks ago
embeddinggemma:300m 8545359ee72 621 MB 8 weeks ago
embeddinggemma:latest 8545259ee72 621 MB 3 months ago
qwen3-coder:latest 86c1097e7fce0 18 GB 3 months ago
+ Desktop ollama run qwen3-coder:latest
>>> write a python code to add 2 numbers
Here's a simple Python code to add two numbers:

'''python
# Method 1: Using input() to get numbers from user
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
result = num1 + num2
print("The sum of (num1) and (num2) is: {result}")

# Method 2: Direct assignment
a = 5
b = 3
sum_result = a + b
print("The sum of (a) and (b) is: {sum_result}")

# Method 3: Using a function
def add_numbers(x, y):
    return x + y

number1 = 10
number2 = 15
total = add_numbers(number1, number2)
print("The sum of (number1) and (number2) is: {total}")
'''
```





The Economics of Local AI

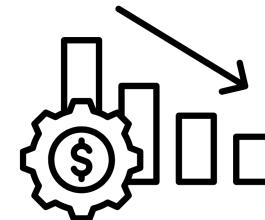
Traditional Cloud SAST:

- Per-developer licensing: \$50-200/month
- Per-scan API costs: \$0.01-0.10 per scan
- Enterprise tier: \$50K-500K/year
- 10-person team annual cost: \$6,000-\$24,000



Local AI Approach:

- Software: \$0 (open source)
- Per-scan cost: \$0 (local compute)
- Hardware: Existing developer machines
- ANY team size annual cost: \$0



The Future Gets Better:

Models improve while cost stays zero



The Future Gets Better, Not More Expensive

Unlike SaaS That Gets More Expensive:

Models Keep Improving

- New open-source models release monthly
- Better quality at same compute cost

Hardware Gets Cheaper

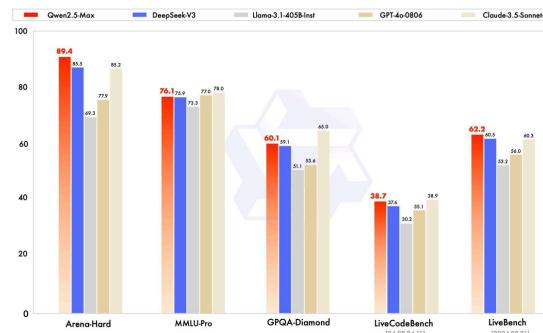
- GPUs that cost \$10K today → \$2K in 3 years
- Same capability, lower barrier to entry

Inference Gets Faster

- Quantization, speculative decoding, flash attention
- Each generation is 2-3x faster

Unlimited Scans Forever

Scan every commit, every branch, every PR — no limits





IMPLEMENTATION

Building the System

Architecture: Hexagonal Design

Ports & Adapters Pattern

Domain Core (Pure Business Logic):

- Security Analyzer — vulnerability detection logic
- Context Assembler — RAG pipeline orchestration
- LLM Service — model interaction abstraction

Infrastructure (Swappable):

- LLM Providers: Ollama, OpenAI, Anthropic
- Vector Stores: ChromaDB, Pinecone, Weaviate
- AST Parsing: Tree-sitter for all languages



Benefits:

Testable • Extensible • Maintainable



Technology Stack

Local LLM Runtime: Ollama

- Easy model management, optimized for local inference
- Recommended: qwen3-coder:30b or deepseek-coder

Vector Database: ChromaDB

- Embedded mode — no server needed
- Persistent storage, fast semantic search

AST Parsing: Tree-sitter

- Battle-tested (VS Code, GitHub)
- Incremental parsing, 40+ language support

Language: Python 3.12+

- Async/await for parallel processing
- Rich ecosystem for ML/AI





FalconEYE: Open Source Implementation

We've Built This: <https://github.com/falconEYE-ai/FalconEYE>

Features:

- All techniques discussed today — implemented and tested
- 9+ language support out of the box
- 4 output formats: Console, JSON, HTML, SARIF
- Production-ready with CI/CD integration

Quick Start:

pip install falconeeye

falconeye index ./your-project

falconeye scan ./your-project

The screenshot shows the GitHub repository page for FalconEYE-ai/FalconEYE. The repository has 4 commits from hardw00t. The README section highlights FalconEYE as a Next-Generation Security Code Analysis Powered by Local LLMs. It mentions that FalconEYE represents a paradigm shift in static code analysis, leveraging large language models to reason about code like a security expert. The repository has 9 stars and 2 forks. It includes sections for Releases (FalconEYE_v2.0), Packages (No packages published), and Contributors (hardw00t, h4ckologic).

Live Demo

Let's See It In Action

Setup (one-time):

```
$ ollama pull qwen3-coder:30b
$ pip install -e .
```

Index Your Codebase:

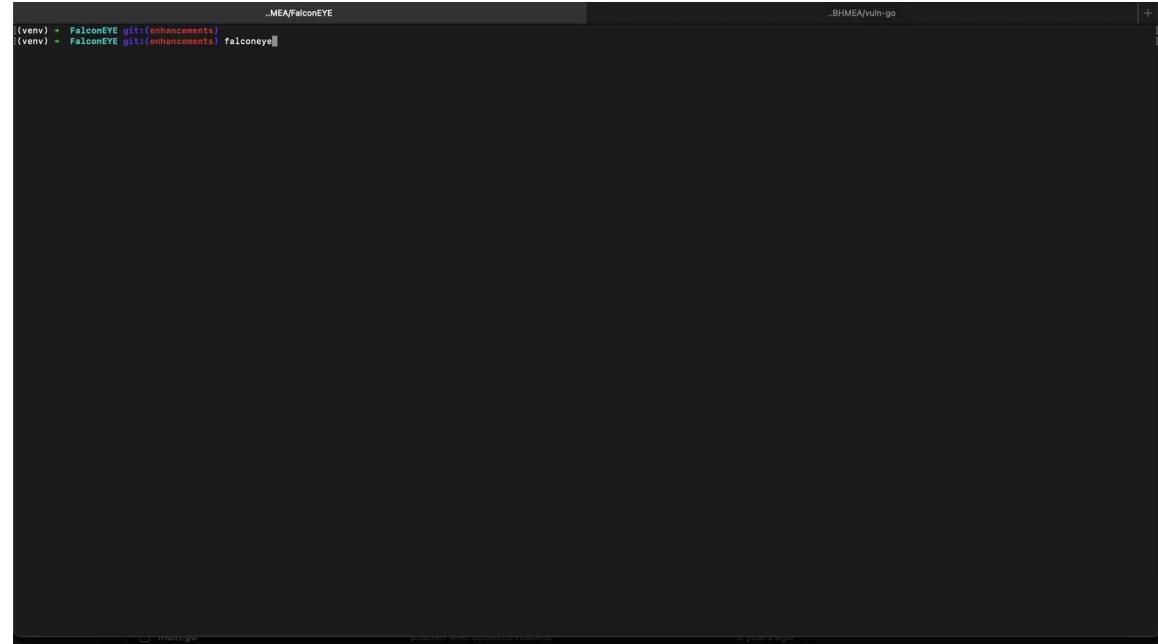
```
$ falconeeye index ./project
```

Run Security Scan:

```
$ falconeeye scan ./project
```

What We'll See:

- Semantic vulnerability detection
- Context-aware analysis
- Actionable fix suggestions



```
..MEA/FalconEYE
(venv) + FalconEYE git:(enhancements) falconeeye
(venv) + FalconEYE git:(enhancements) falconeeye

..BHMEA/vuln-go
```



Roadmap & Future Work

Near Term:

- IDE plugins
- Pre-commit hooks integration
- More language support

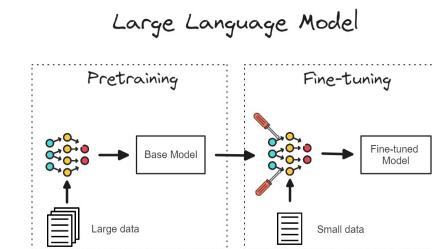


Medium Term:

- Custom model fine-tuning on your codebase
- Team knowledge base integration
- Compliance report generation

Research:

- Multi vulnerability chains
- Automated fix generation
- Security regression testing





IdentityShield
Summit '26

Thank You

Questions? | github.com/falconEYE-ai/FalconEYE



IdentityShield
Summit '26

Any Feedback?



github.com/falconEYE-ai/FalconEYE