AquilaX AI (Securitron) VS Frontier AI Models

A True Positive / False Positive Vulnerability Triage Benchmark

AquilaX AI/ML & Application Security Engineering

August 2026

Executive Summary

This report benchmarks AquilaX's proprietary Securitron AI against five leading general-purpose large language models - DeepSeek, Claude, OpenAI, Kimi, and Grok - on the task every AppSec team cares about most: correctly telling a real vulnerability apart from scanner noise.

All six models classified the same 340 scanner findings, produced by AquilaX's 32-engine scanner suite against three intentionally vulnerable test applications - a Python app, a Java/Maven app, and OWASP Juice Shop (Node.js/TypeScript, plus its Solidity/Web3 and Terraform IaC challenges) - as either a True Positive or a False Positive. Each classification was checked against an independent manual security review, which serves as ground truth (225 confirmed real vulnerabilities and 115 confirmed false positives).

Securitron classified findings correctly **90.6% of the time** - roughly 1.2× the best general-purpose model (Claude, at 75.9%) and nearly 1.4× the weakest (Grok, at 65.9%). The margin is driven overwhelmingly by recall: Securitron caught 94.7% of real vulnerabilities, well ahead of Claude's 70.7% and more than 40 points clear of Grok's 53.8%.

Adding Juice Shop's large, realistic mix of benign findings (115 of the 340) also surfaces a genuine trade-off worth stating plainly: on this bigger, noisier set, Securitron is not the most conservative model. It correctly dismissed 82.6% of benign findings - slightly behind Claude, DeepSeek, and notably Kimi (93.9%) and Grok (89.6%). But Kimi and Grok's caution comes at a steep cost elsewhere: they achieve that precision partly by flagging far less overall, which is exactly why they also miss the most real vulnerabilities (96 and 104 respectively). Securitron's 12 missed real vulnerabilities against 20 false alarms is a markedly better trade than either extreme.

Methodology

Test Corpus

The benchmark uses findings generated by scanning three intentionally vulnerable applications: two AquilaX maintains specifically to validate scanner and triage accuracy (AquilaX-AI/vulnapp-python, AquilaX-AI/vulnapp-java), plus OWASP Juice Shop - the most widely used vulnerable application in the security community. The Python application seeds known, deliberate issues - including SQL injection, OS command injection, hardcoded credentials, missing CSRF protection, missing authentication, debug mode exposure, and a Dockerfile/OpenAPI spec/CI workflow with intentional misconfigurations - alongside a requirements.txt built to trigger real dependency license conflicts and known-CVE advisories. The Java/Maven application adds a second, independent codebase: a pom.xml pulling in deliberately outdated dependencies with known CVEs (Apache Commons Collections insecure deserialization, Google Guava information disclosure, MySQL Connector/J privilege escalation, Netty request-smuggling and DoS issues, Apache HttpClient and Xerces/Xalan XML vulnerabilities, HtmlUnit code execution), plus repository-hygiene gaps (missing LICENSE and SECURITY.md) and a JSP file flagged for AI-generated-code patterns.

OWASP Juice Shop adds a third, much larger and more realistic codebase: a full Node.js/TypeScript/Angular e-commerce application covering the OWASP Top 10, plus two categories the earlier two apps didn't touch - a set of intentionally flawed Solidity/Web3 smart-contract challenges, and a Terraform IaC module with seeded cloud-security misconfigurations (open security groups, disabled IAM Access Analyzer, missing Shield Advanced). Critically, Juice Shop's scale also contributes the bulk of this benchmark's false-positive test cases - 112 of the corpus's 115 confirmed-benign findings come from its size and the number of near-miss, reference, and “corrected” code variants scanners flag against it - giving this round a far more realistic mix of signal and noise than the first two apps alone provided.

AquilaX's 32 parallel scanning engines (SAST, SCA, secrets, IaC, API spec, and container analysis) produced 340 distinct findings across the three codebases combined. Each finding was independently reviewed by a human analyst and labelled True Positive or False Positive; that manual review is the ground truth this benchmark measures every model against.

Evaluation Process

The same 340 findings - CWE classification, affected file, code context, and description - were presented to Securitron and to five general-purpose models (DeepSeek, Claude, OpenAI, Kimi, and Grok), each asked to classify the finding as TRUE_POSITIVE or FALSE_POSITIVE. Every model's verdict was then compared against the manual-review ground truth to compute accuracy, recall on real vulnerabilities (how many true issues each model caught), and specificity (how many genuinely benign findings each model correctly dismissed).

Throughout this report, each model is referred to by its provider name (Securitron, DeepSeek, Claude, OpenAI, Kimi, Grok). The exact version tested for each is listed below for reproducibility:

Report Label	Model Version Tested	Provider
Securitron	Per-customer Review Model (production build)	AquilaX
DeepSeek	DeepSeek V4 Flash	DeepSeek AI
Claude	Claude Sonnet 4.6	Anthropic
OpenAI	GPT-5.5	OpenAI
Kimi	Kimi K3	Moonshot AI
Grok	Grok 3	xAI

Results

The table below summarises how each model performed across all 340 findings.

Model	Overall Accuracy	Real Vulns Caught	Real Vulns Missed	Benign Findings Dismissed Correctly
Securitron	90.6% (308/340)	213 / 225 (94.7%)	12	95 / 115 (82.6%)
Claude	75.9% (258/340)	159 / 225 (70.7%)	66	99 / 115 (86.1%)
DeepSeek	73.8% (251/340)	151 / 225 (67.1%)	74	100 / 115 (87.0%)
OpenAI	70.9% (241/340)	146 / 225 (64.9%)	79	95 / 115 (82.6%)
Kimi	69.7% (237/340)	129 / 225 (57.3%)	96	108 / 115 (93.9%)
Grok	65.9% (224/340)	121 / 225 (53.8%)	104	103 / 115 (89.6%)

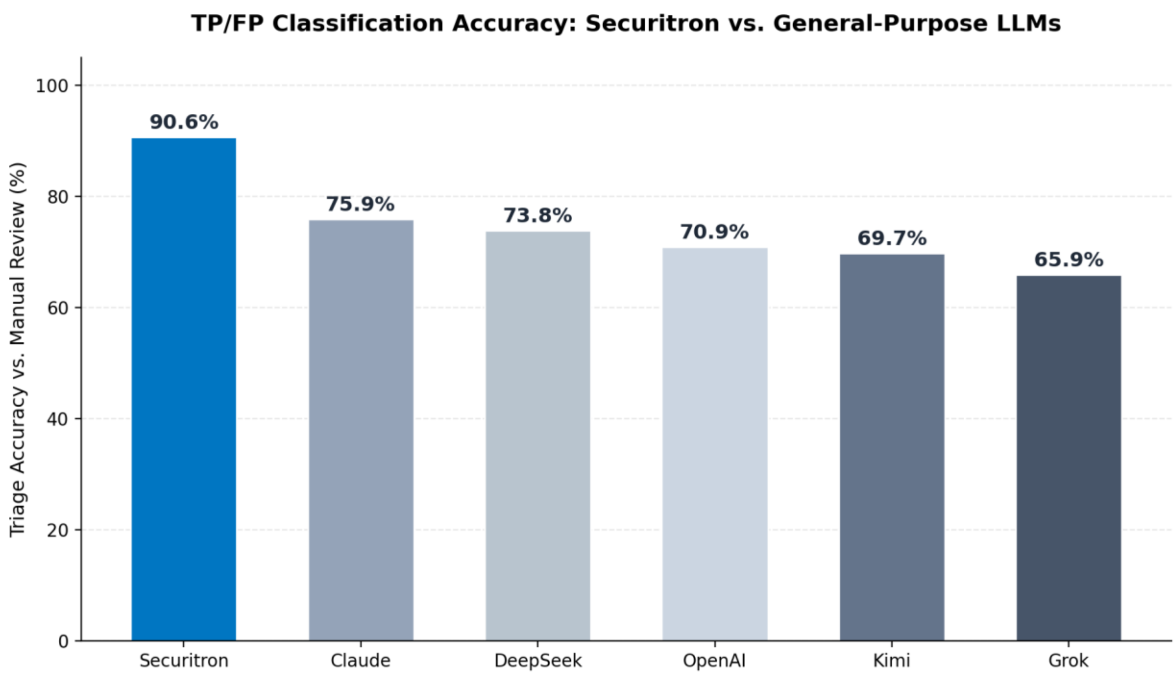

Figure 1. Overall triage accuracy against manual-review ground truth.

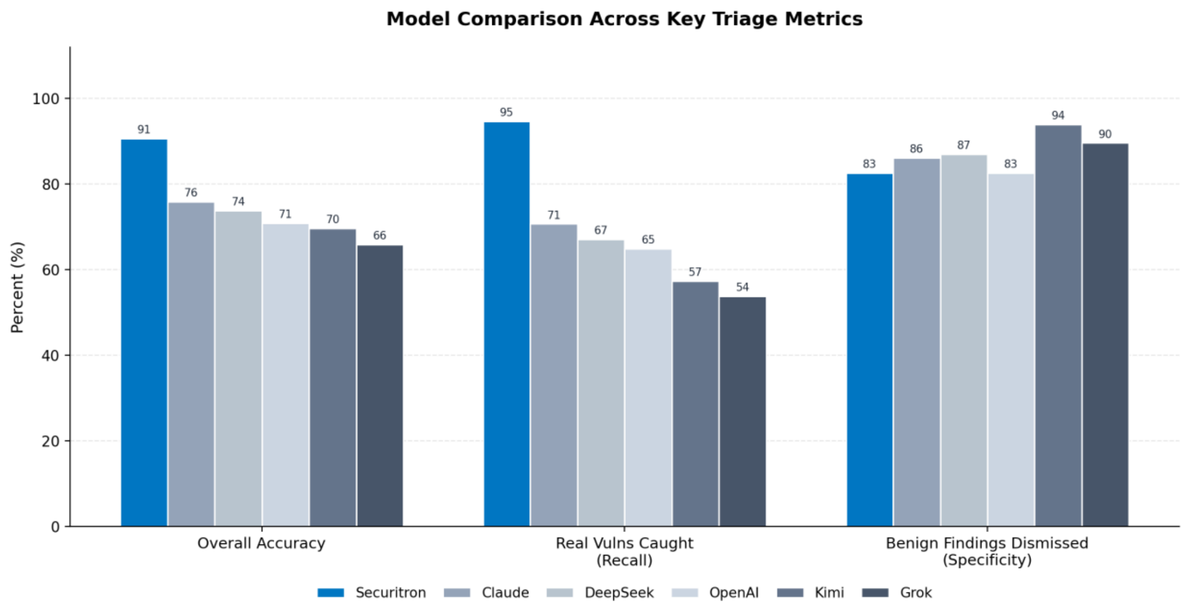

Figure 2. Accuracy, recall on real vulnerabilities, and specificity on benign findings, side by side.

Performance by Finding Category

Splitting the test set into code/config-level findings (SAST, secrets, IaC, Solidity, API spec - 243 findings) and SCA/dependency/repo-policy findings (license and CVE issues, plus hygiene checks - 97 findings) shows where the general-purpose models fall furthest behind:

Model	Code & Config-Level (n=243)	SCA / Dependency & Repo Policy (n=97)
Securitron	89.7%	92.8%
Claude	83.5%	56.7%
DeepSeek	81.9%	53.6%
OpenAI	79.8%	48.5%
Kimi	80.7%	42.3%
Grok	77.4%	37.1%

On code-level vulnerabilities - the injection flaws, hardcoded secrets, access-control gaps, and IaC misconfigurations a security engineer would triage first - Securitron leads at 89.7%, with Claude closest among the general-purpose models at 83.5%; the field is tighter here than in earlier rounds of this benchmark, partly because Juice Shop's large TypeScript codebase plays to patterns every model has seen extensively in training. On dependency and repo-policy findings (SCA license/CVE issues plus hygiene checks like missing LICENSE/SECURITY.md files), the gap widens sharply and holds the pattern from earlier rounds: Grok correctly classified only 37.1% of findings, and even Claude - the strongest general-purpose model in this category - reached just 56.7%, while Securitron held at 92.8%.

Breakdown by Programming Language / File Type

With the addition of OWASP Juice Shop, this benchmark now spans five distinct languages/ecosystems rather than two: Python, Java/Maven, TypeScript/JavaScript (Node.js + Angular), Solidity (smart contracts), and Terraform (IaC):

Model	Python (.py)	Java (Maven/JSP)	TypeScript/JS (Node)	Solidity (Web3)	Terraform (IaC)	Other (Config/Deps)
Securitron	100%	100%	86.8%	98.0%	78.7%	91.8%
Claude	100%	77.8%	94.7%	91.8%	72.1%	50.0%
DeepSeek	100%	66.7%	93.4%	91.8%	68.9%	50.0%
OpenAI	100%	86.1%	93.4%	83.7%	72.1%	34.7%
Kimi	95.0%	80.6%	93.4%	89.8%	72.1%	30.6%
Grok	95.0%	66.7%	89.5%	89.8%	70.5%	26.5%

Securitron still leads on four of the five languages - Java (100%), Solidity (98.0%), Terraform (78.7%), and ties for first on Python (100%) - but this round surfaces a genuine exception worth stating plainly: on TypeScript/JavaScript, Securitron (86.8%) actually trails every general-purpose model, with Claude leading at 94.7%. This tracks with what you'd expect - JavaScript/TypeScript security patterns (XSS, prototype pollution, insecure Express middleware) are some of the most heavily represented examples in general-purpose LLM training data, narrowing Securitron's usual advantage. Its lead widens again sharply on categories with less public training material: Solidity smart-contract logic and Terraform cloud-IaC misconfigurations, where Securitron's domain-specific training still holds a clear edge over general-purpose reasoning.

Representative Findings: The Code Behind the Numbers

A sample of five findings illustrates what these percentages mean in practice - the actual evidence AquilaX's scanners surfaced, and how each model called it against the confirmed ground truth (GT). These span all three codebases and five finding categories, including one genuinely benign finding to illustrate precision as well as recall:

Finding (CWE)	Code / Evidence	GT	Securitron	DeepSeek	Claude	OpenAI	Kimi	Grok
Missing SECURITY.md Policy (CWE-710)	Repository has no SECURITY.md in root, /docs, or /.github	TP	✓	✗	✗	✗	✗	✗
Insecure Deserialization (CWE-502, Java/Maven)	commons-collections 3.2.1 - unsafe deserialization of untrusted data	TP	✓	✓	✓	✓	✓	✗
Reentrancy False Alarm (Solidity/Web3)	withdraw() balance check flagged as reentrancy - not exploitable here	FP	✓	✗	✓	✗	✗	✗
IAM Access Analyzer Disabled (CWE-710, Terraform IaC)	AWS IAM Access Analyzer not enabled in networking.tf	TP	✓	✓	✗	✗	✗	✗
Forbidden License: GPL-3.0 (SCA)	PyQt6 (v6.6.1) uses GPL-3.0-only - forbidden by policy	TP	✓	✓	✓	✗	✗	✓

Securitron is correct on all five, including the one genuinely benign finding (a Solidity balance check that looks like a reentrancy risk but isn't exploitable in this guarded implementation) - a case where DeepSeek, OpenAI, Kimi, and Grok all raised a false alarm and only Claude joined Securitron in correctly dismissing it. On the real vulnerabilities, the general-purpose models are inconsistent in different ways depending on the finding: most catch the well-known Commons Collections deserialization CVE, but only DeepSeek joins Securitron on the less-publicized IAM Access Analyzer misconfiguration. No competing model catches everything here, and none out-performs Securitron on any individual row - which matches the aggregate pattern in the Results table above.

Where Each Model Missed Real Vulnerabilities

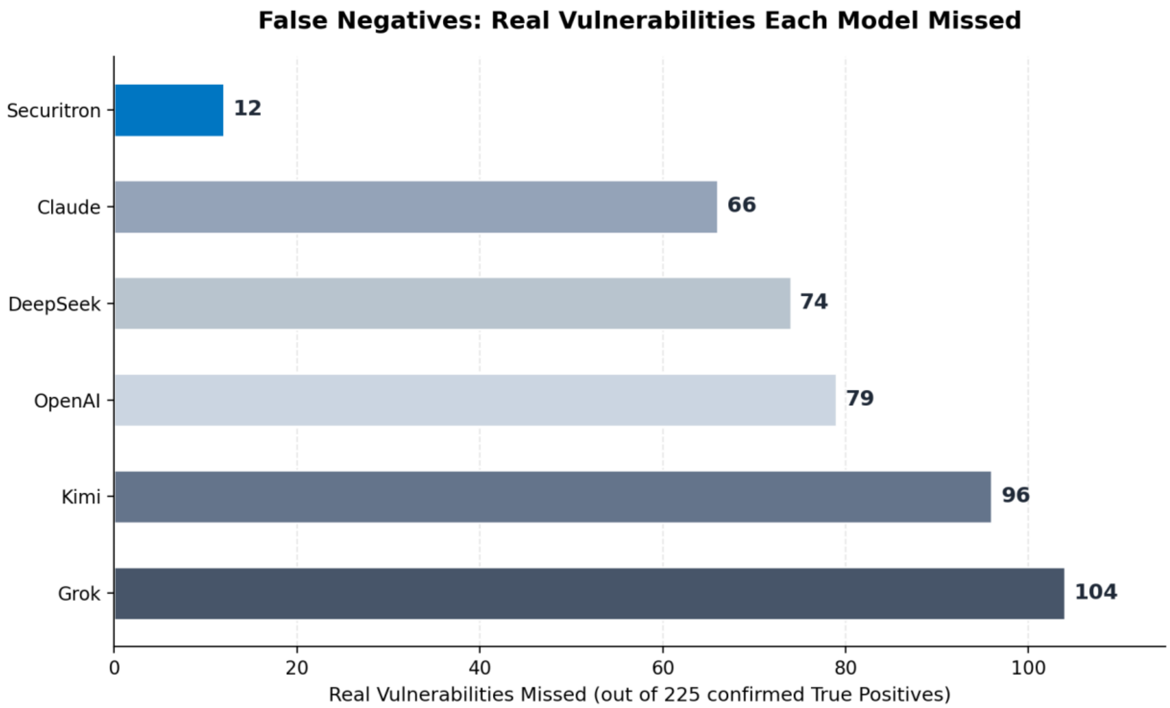

Figure 3. Confirmed real vulnerabilities each model failed to flag, out of 225.

Securitron's 12 misses span all three codebases but stay concentrated in the same pattern as earlier rounds: 7 are the Python app's transitive dependency advisories (a Flask session header nuance, a urllib3 proxy-header edge case, a redirect-following decompression-bomb safeguard), plus a PII exposure finding and a Dockerfile base-image issue. The remaining 4 are new: three TypeScript findings (a hardcoded HMAC signing key, an unsafe vm.runInContext() sandbox escape, and an insecure object-property assignment) and one reflected-XSS gap in a Handlebars view template. It did not miss a single SQL/command injection, authentication, or access-control vulnerability across any of the three codebases, and caught every seeded Java/Maven CVE.

On the other side of the ledger, Securitron's 20 wrongly-flagged benign findings are heavily concentrated in one area: 13 of the 20 are Terraform IaC findings - cloud-configuration checks (Shield Advanced, IAM Access Analyzer, security-group descriptions) where AquilaX's own policy is stricter than what the manual reviewer counted as an active risk in this test environment. The remaining 7 are split across TypeScript (6) and Solidity (1).

Claude was the strongest general-purpose model on recall, missing 66 real vulnerabilities, followed by DeepSeek at 74. Both missed most of the same dependency-license, repo-hygiene, and CVE-advisory findings across the three codebases. OpenAI missed 79, Kimi 96, and Grok missed the most at 104 - in Grok and Kimi's case, that heavier miss rate is the flip side of the higher specificity noted earlier: both models flag far less overall, which reduces false alarms but costs them real catches.

How Securitron Is Trained

Securitron is not a single model - it's a suite of purpose-built AI components, each fine-tuned for a specific step in the triage pipeline, according to AquilaX's public documentation:

Component	Base Model	Adaptation	Role
Review Model	GraphCodeBERT (encoder classifier)	Trained on verified findings, per-customer + cross-org feedback	Classifies every scanner finding as TP/FP with a confidence score
AI Scanner Model	Qwen2.5-Coder-3B-Instruct	LoRA rank 512, 4-bit quantised, 100K files, 5 epochs (SFTTrainer/TRL)	Deep semantic vulnerability analysis on source files
Security Assistant (Q&A)	Qwen2.5-Coder-0.5B	LoRA rank 256, alpha 64	Answers developer questions about findings
Securitron Chat	Chat model, 8192-token context	ChatML format, up to 5-exchange history	Multi-turn conversational triage support
Query Model	FLAN-T5-base	Fine-tuned for NL→SQL	Translates natural-language questions into PostgreSQL queries

Review Model - the TP/FP classifier

The component directly responsible for the triage decisions benchmarked in this report is the Review Model, a GraphCodeBERT-based classifier. It reads each scanner finding together with its code context and classifies it as True Positive or False Positive, and its accuracy improves as it incorporates a customer's own historical feedback alongside patterns learned across AquilaX's broader customer base.

AI Scanner Model - deep semantic analysis

For deeper source-level analysis, AquilaX fine-tunes unsloth/Qwen2.5-Coder-3B-Instruct using Low-Rank Adaptation (LoRA) at rank 512 across the attention and MLP projection layers, with the base model loaded in 4-bit quantisation. Training data is a custom corpus of 100,000 files, each pairing source code with a structured list of the vulnerabilities present (or an empty list where none exist), formatted as conversational prompts in which the system message defines the assistant's role as “Securitron.” Fine-tuning runs for 5 epochs via Hugging Face's SFTTrainer (TRL library), batch size 2, learning rate 2e-4 with an 8-bit AdamW optimiser and cosine annealing, a 5-step warmup, and an 8,192-token maximum sequence length, with gradient checkpointing enabled through Unsloth to manage memory.

Continuous, per-customer learning

Beyond the base training run, Securitron trains a dedicated model per customer on top of the shared foundation, adapting to that codebase's frameworks, patterns, and developer conventions. Every accept/reject decision a security team makes on a finding feeds back into that customer's model, so accuracy compounds sprint over sprint rather than staying fixed at deployment time.

AquilaX reports that, in production across more than 153,000 protected applications, this architecture eliminates roughly 93.5% of scanner false positives while completing triage inside the scan window - broadly consistent with the 90.6% accuracy this benchmark measured across the vulnapp-python, vulnapp-java, and Juice Shop test sets; the more nuanced false-positive rate this larger, noisier corpus surfaces (82.6% correctly dismissed) is discussed under Results above.

Why General-Purpose LLMs Underperform Here

- No security-specific fine-tuning: DeepSeek, Claude, OpenAI, Kimi, and Grok are broad-purpose models without training data curated specifically around vulnerability triage, so they lack the pattern grounding Securitron's 100K-file training set provides.
- No feedback loop: none of the five general-purpose models learn from a team's accept/reject history the way Securitron's per-customer model does - every scan is evaluated from a cold start.
- Weakest on dependency/SCA and repo-policy reasoning: license-compatibility, CVE-applicability, and hygiene checks require matching a specific policy matrix rather than reasoning about code semantics - an area where every general-purpose model scored well below its own code-level number, from 37.1% (Grok) up to 56.7% at best (Claude).
- Inconsistent across languages: no general-purpose model held its Python-level accuracy on the Java/Maven or Terraform corpora - several models drop 20-30 points moving off Python and TypeScript, while Securitron holds 100% on Java and leads on Terraform and Solidity.
- Precision/recall trade-offs cut both ways: Kimi and Grok post the best specificity in this round (93.9% and 89.6%) but only by being far more conservative overall - that same caution is why they also miss the most real vulnerabilities (96 and 104). Securitron and OpenAI both land lower on specificity (82.6%), but only Securitron pairs that with a 94.7% recall; OpenAI's lower precision buys it nothing, since its recall (64.9%) is unremarkable too.

Conclusion

On this benchmark, Securitron outpaces even the strongest general-purpose model tested (Claude, at 75.9%) by close to 15 accuracy points, and outpaces the weakest (Grok, at 65.9%) by close to 25. The advantage is largest exactly where it matters most operationally - correctly catching real, exploitable vulnerabilities, where Securitron's 94.7% recall clears the next-best model by 24 points. The results also surface two honest areas for follow-up: dependency/SCA/repo-policy findings remain Securitron's softest spot relative to its own code-level performance, and on this larger, noisier corpus its precision (82.6% of benign findings correctly dismissed) trails several general-purpose models - a real trade-off, not just a strength, and one worth investigating alongside the TypeScript/JavaScript category where general-purpose models currently hold a narrow edge. Extending the benchmark further - additional languages (Go, Ruby), a larger volume of findings, and repeated runs - would sharpen how well these results generalise.

Source data: TP_FP_Model_Benchmarkspreadsheet - ai_performance.csv. Test targets: AquilaX-AI/vulnapp-python (github.com/AquilaX-AI/vulnapp-python), AquilaX-AI/vulnapp-java (github.com/AquilaX-AI/vulnapp-java), and OWASP Juice Shop (github.com/juice-shop/juice-shop).