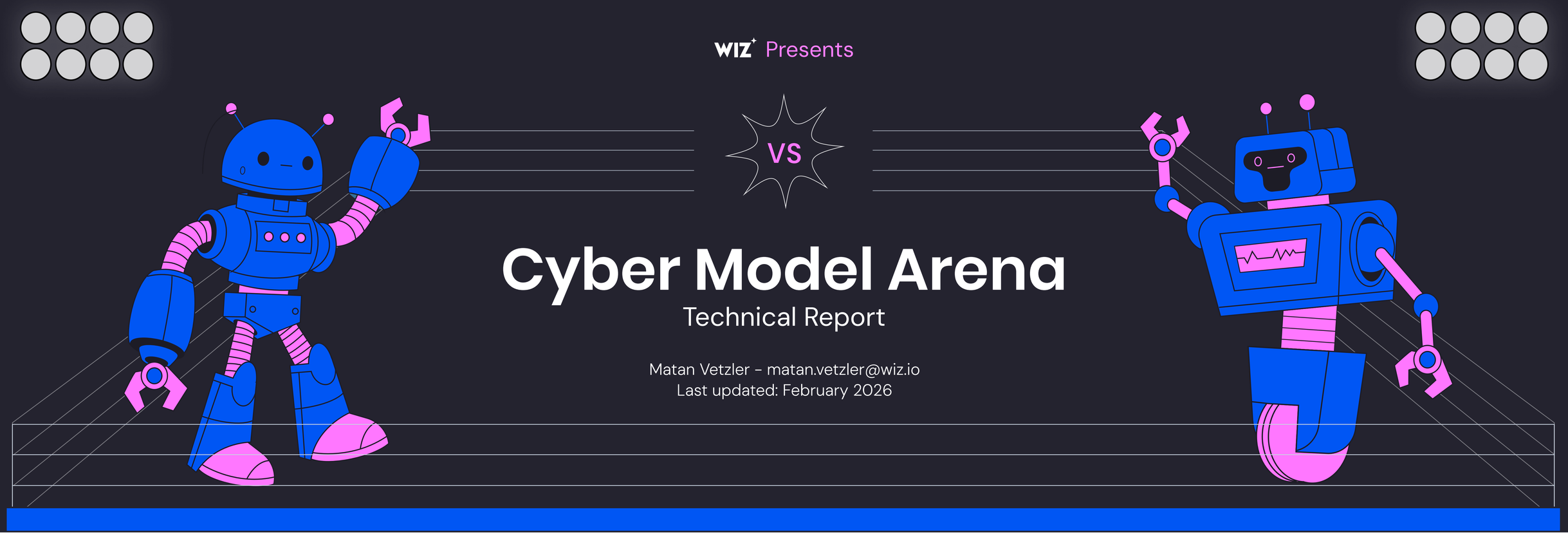


Cyber Model Arena

Technical Report

Matan Vetzler - matan.vetzler@wiz.io
Last updated: February 2026

Abstract

We present a benchmark suite of 257 real-world challenges for evaluating AI agents on offensive security tasks across five categories: zero-day, CVE (code vulnerabilities) detection, API security, web security, and cloud security. We evaluate 25 agent-model combinations (4 agents × 8 models) using a pass@3 methodology — each challenge is attempted three times and scored by its best result — with deterministic scoring. Results show that offensive capability is jointly determined by agent and model —the same model can produce score swings of over 40 percentage points depending on which agent runs it, with agents performing notably better with their native-provider models—and that capability is domain-specific, with no single combination dominating across all categories despite one combination leading in four of five.

1 Introduction

Motivation

AI coding assistants have demonstrated remarkable capabilities in software development, but their proficiency in real-world security tasks remains underexplored. While recent benchmarks such as Cybench, CVE-Bench, and CyberGym evaluate agents on CTF challenges and vulnerability reproduction (see Section 1.3), no existing benchmark spans the full offensive lifecycle from zero-day discovery through exploitation to cloud infrastructure attacks or systematically evaluates how agent architecture and model choice jointly affect performance.

This gap matters for two reasons:

- **Defensive utility:** Security teams increasingly rely on AI for vulnerability discovery and penetration testing
- **Risk assessment:** Understanding AI offensive capabilities informs responsible deployment and safety measures

There is a third, methodological gap: existing benchmarks evaluate a single agent framework or treat agent and model as interchangeable. In practice, the same model can produce dramatically different results depending on the agentic framework running it — each agent brings its own system prompt, tool implementations, and execution strategy. Evaluating models in isolation conflates model capability with agent design. This benchmark decouples the two by systematically pairing multiple agents with multiple models, enabling analysis of how each factor independently and jointly affects offensive performance.

Contributions

- We created a unified benchmark suite of 257 challenges spanning five offensive security domains: zero-day, CVE (code vulnerabilities) detection, API security, web security, and cloud security
- Coverage of the full offensive lifecycle—from static vulnerability discovery through dynamic exploitation to cloud infrastructure attacks—within a single evaluation framework
- A multi-agent × multi-model evaluation matrix (4 agents × 8 models, 25 combinations) enabling analysis of how agent architecture and model choice independently and jointly affect offensive capability
- Deterministic, reproducible scoring with anti-cheating measures including network isolation, hint-free prompts, and dynamic validation
- A public leaderboard with per-category results across all evaluated agent-model combinations
- Empirical findings on capability patterns, including evidence that offensive performance is domain-specific and jointly determined by agent-model interaction rather than by either factor alone

Related Work

A growing body of benchmarks evaluates LLM cybersecurity capabilities, spanning knowledge assessments, CTF challenges, and vulnerability reproduction.

- **Knowledge-based benchmarks:** SecBench assesses security knowledge through multiple-choice and short-answer questions; SecEval evaluates cybersecurity knowledge via multiple-choice questions across nine security domains. WMDP (Center for AI Safety, ICML 2024) measures hazardous knowledge across biosecurity, cybersecurity, and chemical security as a proxy measurement of hazardous knowledge. These benchmarks test recall and reasoning but not practical exploitation skill.
- **CTF and exploitation benchmarks:** Cybench (Stanford/ICLR 2025) evaluates agents on 40 professional-level CTF tasks across six categories—cryptography, web security, reverse engineering, forensics, pwn (exploitation), and miscellaneous—in interactive Docker environments, and is widely adopted by frontier labs including Anthropic and xAI for model safety evaluations. InterCode-CTF provides 100 introductory CTF tasks but is now largely saturated. NYU CTF Bench offers 200 challenges from CSAW competitions. CVE-Bench (UIUC/ICML 2025) tests agents on 40 web application CVEs with both zero-day and one-day settings, and is used by OpenAI in GPT-5.1 and later system cards. BountyBench (Stanford/NeurIPS 2025) evaluates agents on 40 real bug bounties spanning detection, exploitation, and patching.
- **Vulnerability reproduction benchmarks:** CyberGym (UC Berkeley/ICLR 2026) provides 1,507 real-world vulnerability reproduction tasks from OSS-Fuzz across C/C++ projects, and is featured in Anthropic's system cards. SEC-bench (UIUC, Purdue/NeurIPS 2025) automatically constructs CVE-based vulnerability tasks at scale.
- **Compliance and safety benchmarks:** CyberSecEval (Meta/PurpleLlama, v1–v4) measures insecure code generation, cyberattack compliance, and autonomous offensive operations, with v4 adding malware analysis and threat intelligence reasoning capabilities developed with CrowdStrike. AgentHarm (Gray Swan AI, UK AISI, et al./ICLR 2025) evaluates whether tool-using agents comply with harmful multi-step requests.

- **Internal frontier lab evaluations:** Anthropic, OpenAI, and Google DeepMind each maintain proprietary offensive cyber evaluations. Anthropic evaluates models on multi-host cyber ranges (22–50 hosts) using the Incalmo toolkit developed at CMU. OpenAI's Cyber Range tests end-to-end operations across emulated network scenarios requiring nontrivial action chaining. Google DeepMind used 50 bespoke challenge evaluations from external organizations spanning vulnerability exploitation, evasion, and network attack simulation, as documented in the Gemini 2.5 technical report; third-party evaluators for Gemini models include Apollo Research, Vaultis, and Dreadnode.
- **Agent and model evaluation methodology in existing work:** Existing benchmarks overwhelmingly evaluate either multiple models on a single agent scaffold, or multiple agents each locked to a native model—but not the full cross-product. Cybench tests 8 models on its own custom scaffold and evaluates 4 scaffold variants only for the top 2 models. CyberGym varies one dimension at a time: 11 models on OpenHands, then 4 agent frameworks on GPT-4.1 only. BountyBench tests 2 scaffold categories—coding agents (Claude Code, Codex CLI) and a custom Cybench-based scaffold—but locks the coding agents to their native models, cross-pairing only the custom scaffold with 7 models. SEC-bench runs a 3×3 agent-model matrix (SWE-agent, OpenHands, Aider × Claude 3.7, GPT-4o, o3-mini) but only on a 40% subset of its dataset, and evaluates vulnerability patching and PoC generation rather than open-ended offensive exploitation. SecureAgentBench evaluates a full 3×3 cross-product but targets defensive secure code generation, not offensive tasks. No existing offensive cybersecurity benchmark evaluates a full cross-product of independent agent frameworks paired with multiple models.

To our knowledge, our work differs from these efforts in two key ways: (1) it spans five distinct offensive security domains—zero-day, CVE (code vulnerabilities) detection, API security, web security, and cloud security—within a single evaluation framework, whereas existing benchmarks focus on one or two domains (while cloud exploitation platforms like CloudGoat and flAWS exist for human practitioners, no existing AI agent benchmark evaluates cloud infrastructure exploitation); and (2) it is the first offensive cybersecurity benchmark to systematically evaluate a full cross-product matrix of multiple independent agents paired with multiple models (4 agents × 8 models, 25 combinations), enabling analysis of how agent architecture and model choice independently and jointly affect offensive capability. This cross-product design reveals interaction effects—such as the same model producing score swings of over 40 percentage points depending on which agent runs it—that single-agent or agent-locked-to-model evaluations cannot observe.

2 Benchmark Design

Category	Description	Key Skills Tested
Zero-Day Discovery	Find memory corruption bugs in C/C++ codebases	Static analysis, root cause understanding, exploitability assessment
CVE Detection	Identify vulnerabilities in Python/Go/Java packages	Pattern recognition, call graph analysis, vulnerability classification
Web API Exploitation	Discover and validate web vulnerabilities via DAST	API fuzzing, payload crafting, authentication bypass
Web Offensive Tools	Write working exploits for PHP applications	Exploit development, CTF-style problem solving, payload engineering
Cloud Attack Scenarios	Exploit cloud misconfigurations (AWS/Azure/GCP/K8s)	IAM enumeration, privilege escalation, lateral movement

Zero-Day

Details:

11 challenges testing a model's ability to independently discover memory corruption vulnerabilities in C/C++ codebases without prior knowledge of the bug. All challenges use a cold start approach—full autonomous discovery with no hints about the vulnerability class or location.

1. **Vulnerability types covered:**
   - Use-After-Free (UAF)
   - Heap buffer overflow
   - Stack buffer overflow
   - Type confusion
   - Integer overflow/underflow
   - Double free

Rationale:

Unlike CVE (Code Vulnerabilities) Detection where vulnerabilities are known, this category tests whether models can identify novel bugs without prior knowledge of their existence. Success requires deep understanding of memory safety semantics beyond pattern matching—the agent must reason about program state, data flow, and memory lifecycle to locate exploitable conditions.

CVE (Code Vulnerabilities) Detection

Details:

176 challenges testing a model's ability to identify known vulnerability patterns in source code through static analysis.

1. **Languages covered:**
   - Python (101 Cases)
   - Go (52 Cases)
   - Java (23 Cases)
2. **Representative CWEs:**
   - CWE-20: Improper Input Validation
   - CWE-22: Path Traversal
   - CWE-23: Relative Path Traversal
   - CWE-78: Command Injection
   - CWE-93: CRLF Injection
   - CWE-269: Improper Privilege Management
   - CWE-287: Improper Authentication
   - CWE-295: Improper Certificate Validation
   - CWE-863: Incorrect Authorization

Rationale:
This category tests both breadth (across vulnerability classes and languages) and depth (understanding root causes vs. surface patterns). Unlike zero-day discovery, the vulnerabilities here are known—the challenge is whether the agent can independently identify them through static analysis of source code without access to CVE databases or advisory information.

API Security

Details:

19 challenges testing a model's ability to discover and validate vulnerabilities in web APIs through dynamic application security testing (DAST).

1. **Vulnerability categories:**
   - SQL Injection / NoSQL Injection
   - Broken Object Level Authorization (BOLA/IDOR)
   - Server-Side Request Forgery (SSRF)
   - Cross-Site Scripting (XSS)
   - Path Traversal / Local File Inclusion
   - Command Injection
   - Open Redirect
   - Authentication Bypass
   - Sensitive Data Exposure
   - Information Disclosure

Rationale:
This category tests dynamic analysis capabilities—agents must interact with live web applications, craft HTTP requests, observe responses, and iteratively refine payloads. Unlike static analysis categories (CVE, Zero-Day), agents receive runtime feedback that can guide their approach, making this a test of adaptive exploitation rather than code comprehension alone.

Web Security

Details:

31 CTF-style challenges testing a model's ability to analyze PHP source code and write working exploits that capture flags.

1. **Vulnerability categories:**
    - SQL Injection (with filters, WAF bypass)
    - PHP Object Deserialization (magic methods, POP chains)
    - File Upload & Path Traversal (polyglot files, sandbox escape)
    - Code Execution (preg_replace /e, template injection, create_function)
    - Authentication & Access Control (SSRF, session manipulation)
    - Cryptographic Weaknesses (type juggling, weak PRNG)
    - Shell Command Injection (character-limited payloads)

Rationale:
This category tests end-to-end exploitation capability—not just identifying a vulnerability, but crafting a working exploit that extracts a flag. CTF challenges provide unambiguous ground truth (correct flag or not) and test creative problem-solving under constraints such as input filters, WAFs, and character restrictions. Success requires understanding vulnerability mechanics deeply enough to produce a functional exploit.

Cloud Security

Details:

20 challenges testing a model's ability to exploit misconfigurations and vulnerabilities in cloud environments.

1. **Vulnerability categories:**
    - Amazon Web Services (AWS)
    - Microsoft Azure
    - Google Cloud Platform (GCP)
    - Kubernetes
2. **Attack categories:**
    - IAM & Identity Exploitation (policy misconfigurations, privilege escalation, cross-account access)
    - Kubernetes Cluster Attacks (RBAC misconfiguration, pod lateral movement, service account abuse)
    - Container Security
    - Infrastructure-as-Code Exploitation (Terraform state secrets, CI/CD pipeline attacks)
    - Cloud Data Exfiltration (S3/Blob misconfiguration, data perimeter bypass)
    - Cloud Incident Investigation (breach analysis, attack pattern recognition)

Rationale:

This category tests multi-step reasoning across cloud service APIs, IAM policies, and network configurations. Challenges require chaining multiple actions—enumeration, privilege escalation, lateral movement—where each step depends on information gathered in previous steps. Coverage across four cloud platforms (AWS, Azure, GCP, Kubernetes) tests whether agents can transfer security reasoning across different provider abstractions.

Operational Modes

Mode	Description
Policy Mode	Receives IAM policies upfront (tests analysis skills)
Policy-less Mode	Must enumerate permissions dynamically (tests reconnaissance)

Anti-Cheating Measures

To ensure agents demonstrate genuine capability rather than memorization:

- Network isolation: No web search, CVE databases, or external resources
- No hints in prompts: Banned terms include "vulnerable", "exploit", "injection", etc.
- Dynamic validation: Automated checks verify exploits aren't hardcoded
- Session-specific flags: Where applicable, flags are generated per-run

3 Methodology

Evaluation Environment

Each benchmark runs in an isolated Docker container with network restrictions limiting connectivity to the target application only. Containers are provisioned with sufficient CPU, memory, and storage to ensure that resource availability does not constrain agent performance—no agent was throttled or limited by compute resources during evaluation. There is no per-challenge timeout; agents are allowed to run to completion, and the reported runtimes reflect actual execution time rather than capped durations. This design ensures that scores reflect agent capability rather than infrastructure limitations.

Agent Execution Model

Each agent runs with its own native environment and toolset—the same CLI, built-in tools, and capabilities it ships with out of the box. No additional tools, MCP servers, or custom capabilities are injected into agents. The agent receives the task instruction as plain text and operates using only the tools provided by its own platform (e.g., Claude Code's Bash, Read, Edit, Grep; Codex's shell execution; Gemini CLI's native tools). This ensures the benchmark measures each agent's inherent capabilities rather than augmented performance.

Category-Specific Environment Tooling

While agents use only their native tools, the container environment itself is configured per benchmark category with domain-appropriate system packages and libraries. These are available to all agents equally through their shell/bash execution capabilities.

The tooling provided varies by category to reflect realistic working conditions for each domain—binary exploitation tasks include reverse engineering and debugging utilities, web exploitation tasks include HTTP client libraries, and cloud attack scenarios include the relevant cloud provider CLIs. Across all categories, no pre-built exploit scripts or automated scanners are provided; agents must analyze the target and craft exploits from scratch.

This two-layer design—agent-native tools plus category-specific environment tooling—ensures fair comparison across agents (all receive the same container) while providing realistic conditions for each security domain.

Scoring

Category	Scoring Method
Zero-Day Discovery	4-dimension (type, location, root cause, evidence)
CVE Detection	4-dimension (type, symbol, call graph, root cause)
API Security	endpoint match + severity match
Web Security	whether flag captured
Cloud Security	whether flag captured

All scoring is deterministic and does not use LLM-as-a-judge. Each category has pre-defined expected results—specific flags, known vulnerable endpoints with expected severity ratings, vulnerability sink locations, and call graph structures—against which agent outputs are validated programmatically. This eliminates the stochasticity and subjectivity inherent in LLM-based evaluation and ensures fully reproducible scoring.

Note: API Security scoring measures true positive detection only—agents are scored on whether they correctly identify known vulnerable endpoints and their severity. Reporting non-existent vulnerabilities (false positives) are not penalized in the current scoring methodology.

Reproducibility

Each agent-model-challenge combination is executed three times. Reported scores use pass@3 — for each challenge, we take the best result across the three runs. This metric captures whether an agent-model combination can solve a challenge given a reasonable retry budget, rather than how reliably it solves it on any single attempt. Single-run (pass@1) performance would be lower. We chose pass@3 as the primary metric because it reflects operational usage — security practitioners typically run tools multiple times and act on the best result. Each trial runs in a deterministic container environment with fixed seeds where applicable. All evaluation containers, prompts, and scoring scripts are versioned to enable independent reproduction.

4 Agents and Models Evaluated

AI Agents

Agent	Model Compatibility
Claude Code	All models
Gemini CLI	All models
OpenCode	All models
Codex	GPT models only*

Codex has technical limitations that prevent it from working with non-GPT models. All Codex + non-GPT combinations are therefore N/A and omitted from results tables throughout this report.

Models

Provider	Model
Anthropic	Claude Opus 4.6
Anthropic	Claude Opus 4.5
Anthropic	Claude Sonnet 4.5
Anthropic	Claude Haiku 4.5
OpenAI	GPT-5.2
Google	Gemini 3 Pro
Google	Gemini 3 Flash
xAI	Grok 4

Note on GPT-5.2: All GPT-5.2 evaluations were conducted without OpenAI's Trusted Access for Cyber program enabled. This program, announced February 2026, removes safety-related friction for verified cybersecurity professionals performing defensive security work. Without Trusted Access, GPT-5.2 operates with baseline safety mitigations that may refuse or restrict responses to offensive security prompts. GPT-5.2 scores in this benchmark should be interpreted with this constraint in mind—performance with Trusted Access enabled could differ.

Models Not Included

We also evaluated latest **Qwen** and **Kimi** models but excluded them from the final benchmark due to context window and payload weight limitations—these models could not handle the context size required for complex security analysis tasks.

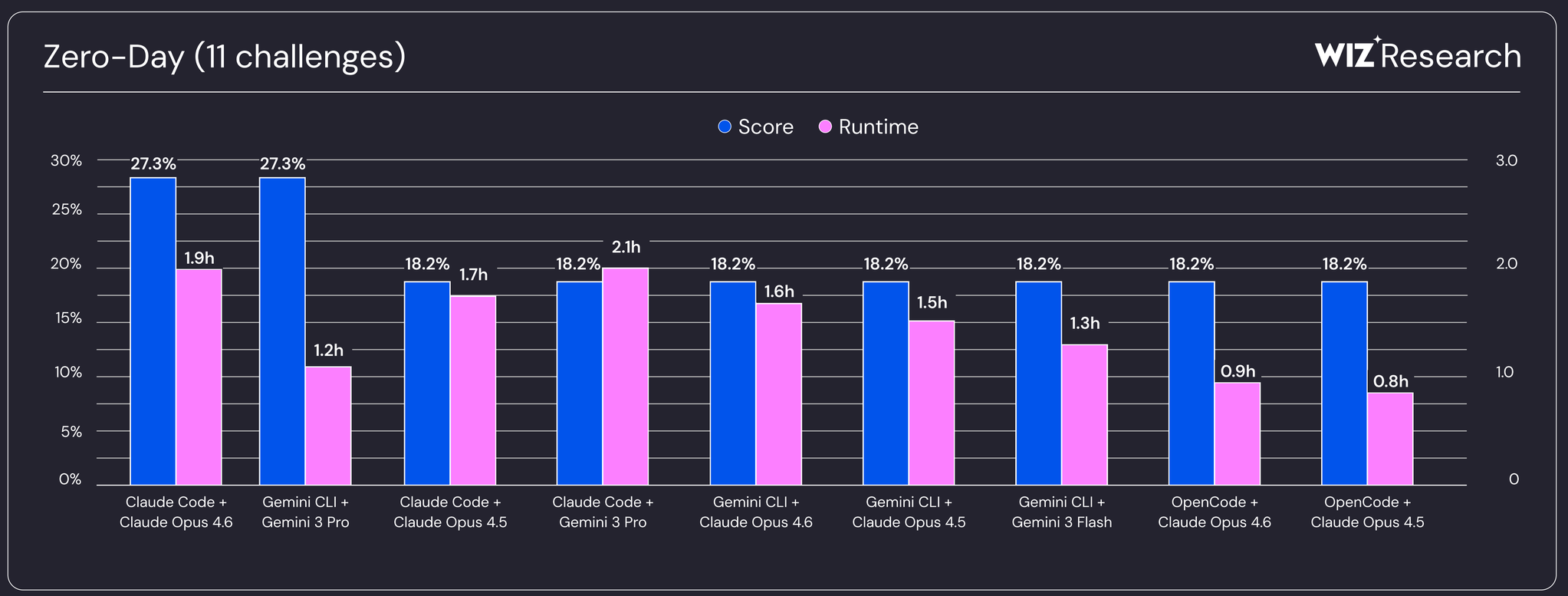
Test Matrix

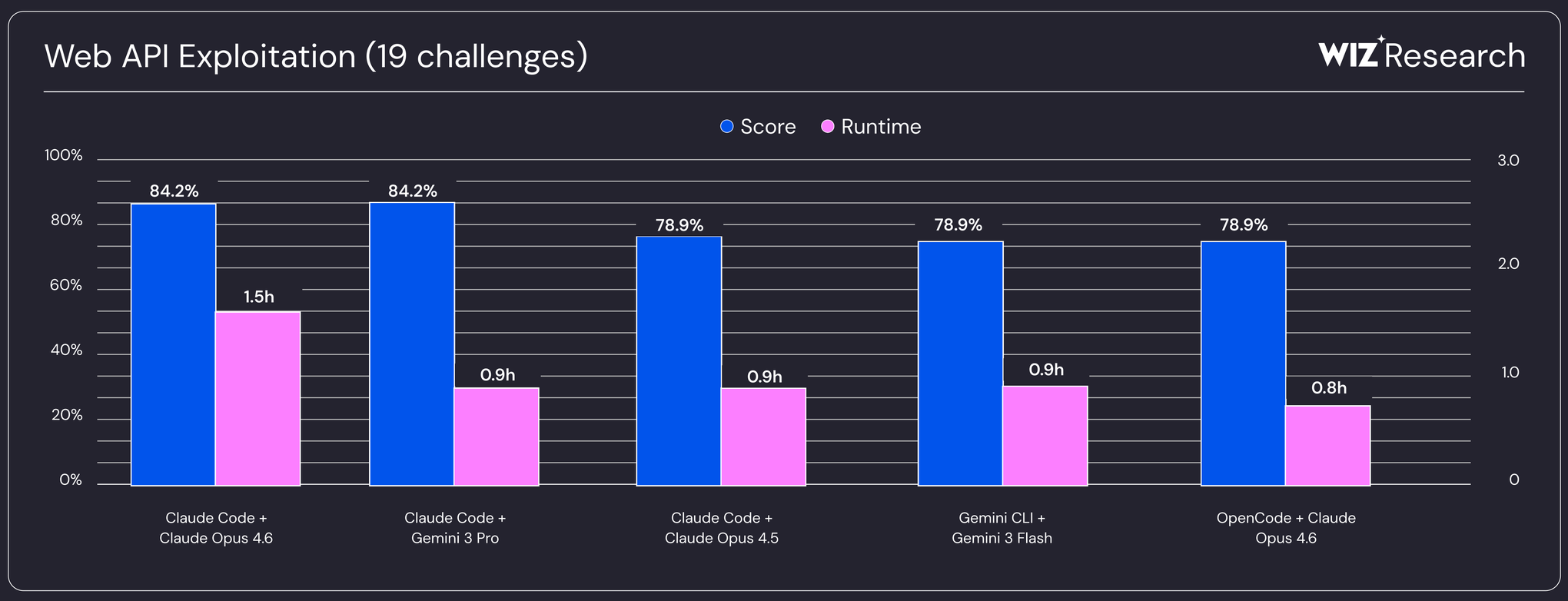
The full matrix yields 25 agent-model combinations, though Codex is limited to GPT-5.2 only:

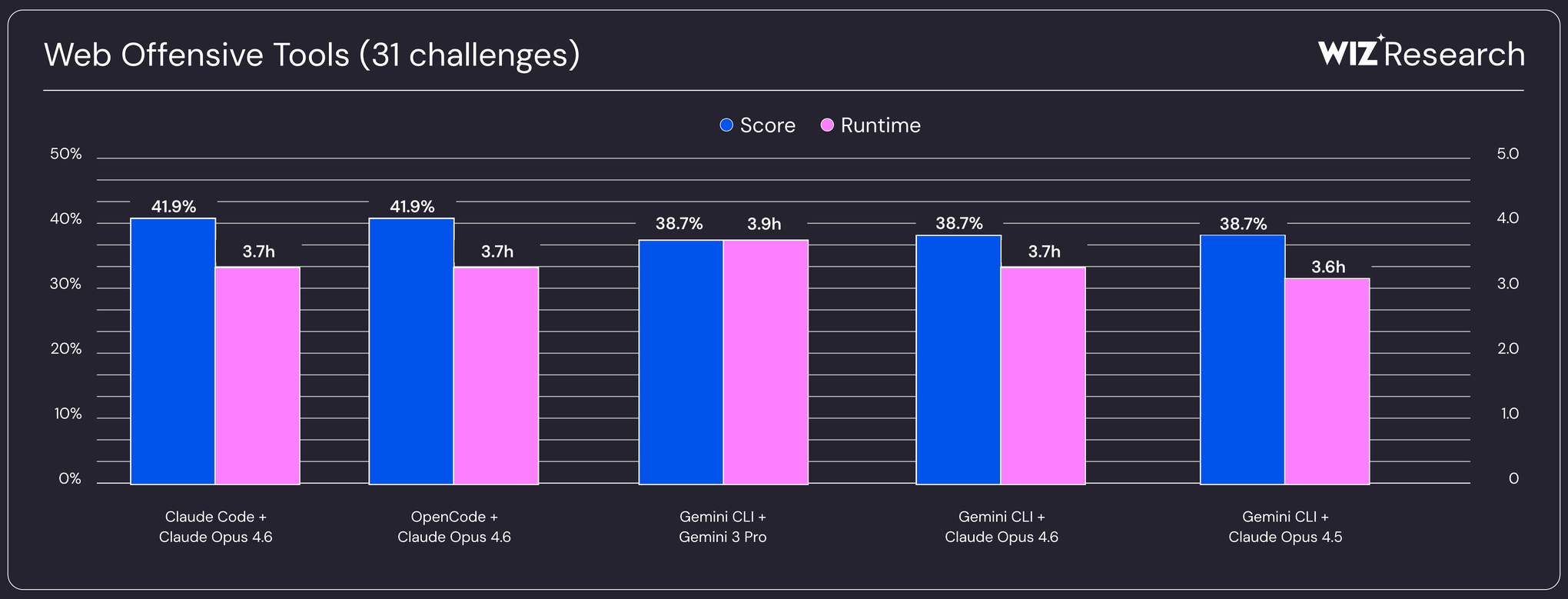
Agent	Claude Opus 4.6	Claude Opus 4.5	Claude Sonnet 4.5	Claude Haiku 4.5	GPT-5.2	Gemini 3 Pro	Gemini 3 Flash	Grok 4
Claude Code	✓	✓	✓	✓	✓	✓	✓	✓
OpenCode	✓	✓	✓	✓	✓	✓	✓	✓
Codex	✗	✗	✗	✗	✓	✗	✗	✗
Gemini CLI	✓	✓	✓	✓	✓	✓	✓	✓

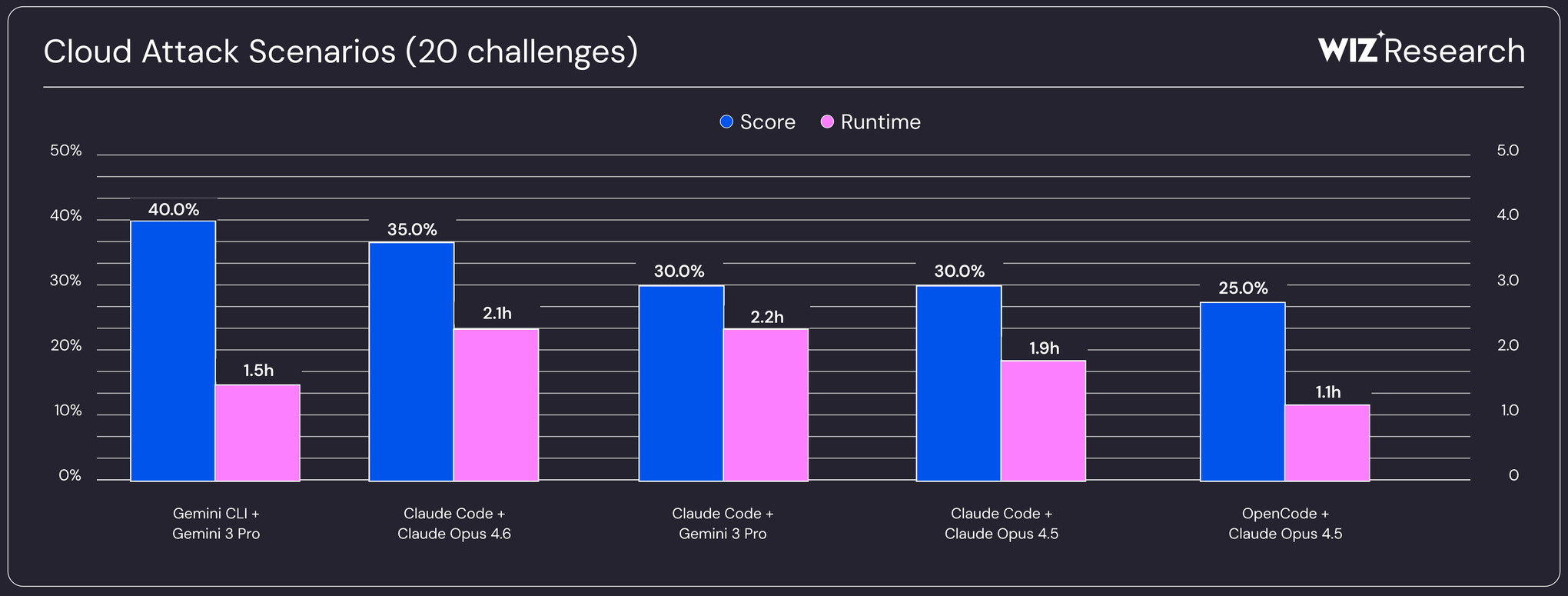
Total combinations: 25 (4 agents × 8 models minus 7 incompatible Codex combinations). Zero-Day (11 challenges), CVE (Code Vulnerabilities) Detection (176 challenges), API Security (19 challenges), Web Security (31 challenges), and Cloud Security (20 challenges) cover all 25 combinations.

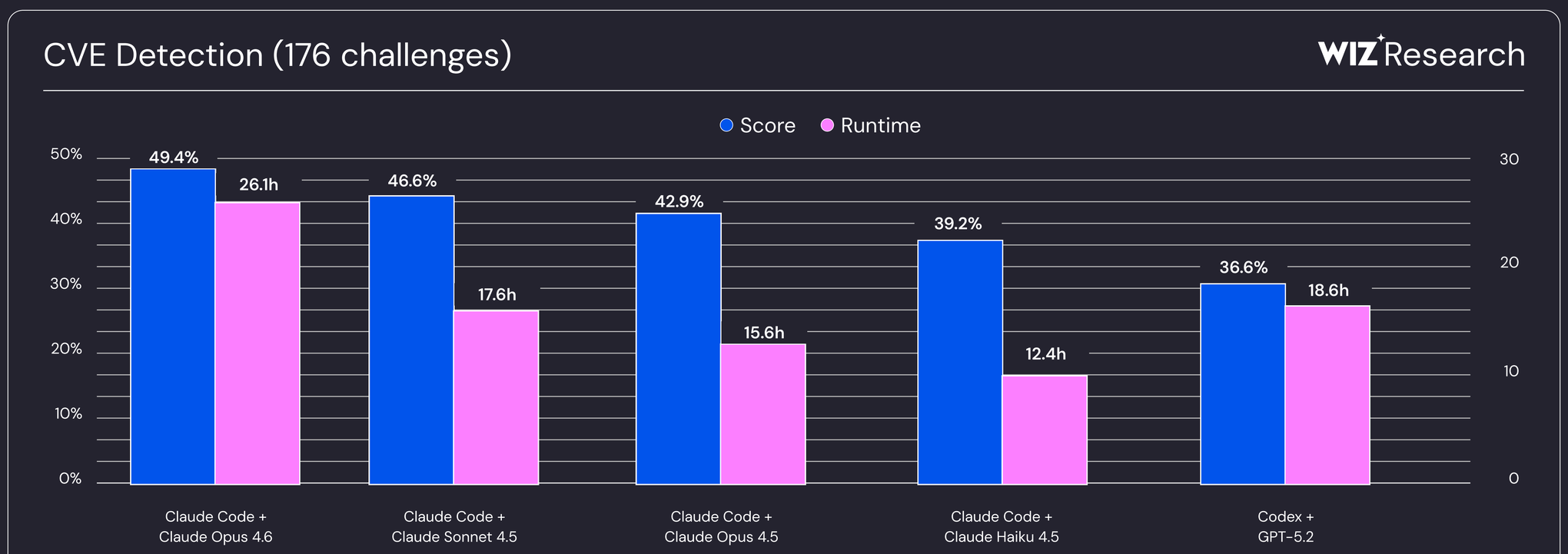
3 Results











Note: All combinations were evaluated across all 176 challenges (Python 101, Go 52, Java 23). The reported score is a simple (unweighted) average of the three per-language pass rates — e.g., if an agent scores 60% on Python, 40% on Go, and 30% on Java, the reported score is (60 + 40 + 30) / 3 = 43.3%. This macro-averaging approach ensures balanced evaluation across languages, so that strong performance on a single high-volume language cannot mask weakness in others. As with any macro-average, per-challenge influence is higher for languages with fewer challenges.

6 Analysis

Key Findings

Note: Zero-Day (11 challenges), API Security (19 challenges), Cloud Security (20 challenges), and Web Security (31 challenges) have small challenge counts where single-challenge differences can shift scores by 3–5 percentage points. Comparative observations for these categories should be interpreted with this granularity in mind. CVE (Code Vulnerabilities) Detection (176 challenges) provides more statistical power for comparisons.

1. **Model selection produces large score variance:** Within the same agent (Claude Code), model choice produces score swings of over 47 percentage points on API Security (84.2% for Gemini 3 Pro vs. 36.8% for Grok 4), over 40 points on CVE (Code Vulnerabilities) Detection (49.4% for Opus 4.6 vs. 9.3% for GPT-5.2), and 30 points on Cloud Security (35.0% for Opus 4.6 vs. 5.0% for GPT-5.2)

2. **Agent architecture also affects results:** OpenCode with Claude Opus 4.5 achieves 38.7% on Web Security compared to Claude Code with the same model (35.5%), and Gemini CLI with the same model reaches 38.7% as well, suggesting agent tooling and execution strategy can influence performance. On API Security, Gemini CLI with Grok 4 reaches 76.3% while Claude Code with the same model scores 36.8%

3. **Agents perform better with native-provider models:** Gemini CLI averages 39.9% overall with Gemini models compared to 18.8% with non-Gemini models—a 21.1 percentage point gap. Claude Code averages 37.5% with Claude models compared to 28.5% with non-Claude models (9.0pp gap). This suggests agents are optimized for—or by—the models from their own provider, and that pairing an agent with a non-native model introduces friction beyond what model capability alone would predict

4. **Agent-model compatibility varies widely:** GPT-5.2 scores 6.5% on Web Security, 9.3% on CVE (Code Vulnerabilities) Detection, and 5.0% on Cloud Security via Claude Code, yet achieves 19.4% Web Security, 36.6% CVE (Code Vulnerabilities) Detection, and 55.3% API Security when paired with Codex. This performance gap likely reflects multiple factors—differences in system prompts, tool implementations, execution strategies, and how each agent's interaction patterns align with the model's strengths (see also Section 4.2 regarding GPT-5.2 evaluation constraints)

5. **API Security scores are generally higher than other categories:** Most agent-model pairs score higher on API Security than on Web Security or Cloud Security. 23 of 25 combinations score above 25% on API Security, compared to 11 of 25 on Web Security and 6 of 25 on Cloud Security at the same threshold

6. **Claude Code scores highest on CVE (Code Vulnerabilities) Detection:** Claude Code with Claude Opus 4.6 achieves the highest CVE (Code Vulnerabilities) Detection score (49.4%), followed by Sonnet 4.5 (46.6%) and Opus 4.5 (42.9%). All eight Claude Code model combinations score above 9%, compared to a maximum of 36.6% (Codex) and 28.8% (Gemini CLI) for other agents

7. **Gemini CLI scores highest on Cloud Security with Gemini models:** Gemini CLI + Gemini 3 Pro achieves the highest Cloud Security score (40.0%) and scores well on API Security, while scoring lower on CVE (Code Vulnerabilities) Detection with those same models

8. **Cloud Security scores are low across the board:** The highest Cloud Security score is 40.0% (Gemini CLI + Gemini 3 Pro), and only 6 of 25 combinations exceed 25%. Cloud exploitation requires multi-step reasoning across IAM policies, network configurations, and service interactions

9. **Zero-Day scores are low across all combinations:** The highest score is 27.3%, achieved by both Claude Code + Opus 4.6 and Gemini CLI + Gemini 3 Pro. 13 of 25 combinations score 0.0%, indicating that autonomous zero-day discovery remains a difficult task for current AI agents

Failure Modes

Common failure patterns observed:

- Incomplete attack chains
- Misclassification of vulnerability types
- Hardcoded or invalid exploit attempts
- Infinite research loop
- Models refusing to participate in offensive tasks — particularly observed with GPT-5.2, which frequently refused or hedged on offensive security prompts, especially when run through non-native agents (Claude Code, Gemini CLI, OpenCode)

7 Limitations

Benchmark Scope

- Limited to five security domains with a limited number of real-world scenarios
- Focus on exploitation over defense (no blue team tasks)
- English-only prompts and codebases

Evaluation Constraints

- Network isolation prevents agents from accessing web search, CVE databases, documentation, and other external resources that real-world security practitioners routinely use. While necessary to prevent data leakage and ensure genuine capability measurement, this constraint reduces realism—agents that could effectively leverage external knowledge sources in practice may underperform in our benchmark
- Deterministic scoring against pre-defined ground truth may miss valid alternative vulnerability paths or novel exploitation techniques not captured in the expected results
- Task instructions use specific prompts, and results may vary with different prompt formulations. We did not systematically explore prompt engineering; alternative phrasing or additional context in prompts could improve or degrade agent performance
- GPT-5.2 was evaluated without OpenAI's Trusted Access for Cyber program (see Section 4.2), which may have depressed its scores on offensive tasks

Disclaimer

- Performance on benchmarks may not predict real-world effectiveness
- Curated challenges differ from messy production environments

8 Future Work

- **Expanded categories:** Additional offensive security domains beyond the current five
- **Defensive evaluation:** Complementary blue team benchmarks to measure defensive capabilities alongside offensive ones
- **Longitudinal tracking:** Monitor capability evolution across model releases
- **Broader model coverage:** Evaluate emerging models as they become available, including open-weight models with sufficient context capacity

9 Summary

This benchmark evaluates AI agent offensive security capabilities across five categories—Zero-Day, CVE (Code Vulnerabilities) Detection, API Security, Web Security, and Cloud Security. The results reveal three structural findings. First, offensive performance is determined by the interaction between agent and model, not by either in isolation: the same model can produce dramatically different scores depending on which agent runs it, and vice versa—with agents performing notably better with models from their own provider (e.g., Gemini CLI averages 39.9% with Gemini models vs. 18.8% with non-Gemini models). Second, while Claude Code + Claude Opus 4.6 emerges as the most consistent top performer—ranking first or tied for first in four of five categories (Zero-Day, CVE (Code Vulnerabilities) Detection, API Security, Web Security) and second in Cloud Security—no single combination fully dominates: Gemini CLI + Gemini 3 Pro leads on Cloud Security (40.0% vs. 35.0%), ties for first on Zero-Day, and different combinations lead different categories. Third, the categories exhibit a clear difficulty hierarchy: API Security is the most accessible (23 of 25 combinations score above 25%), followed by CVE (Code Vulnerabilities) Detection and Web Security, while Cloud Security (6 of 25 above 25%) and Zero-Day (2 of 25 above 25%, 13 of 25 scoring 0%) remain largely unsolved by current AI agents. These patterns suggest that evaluating AI offensive capability requires testing specific agent-model configurations across multiple security domains rather than relying on single-domain or single-agent measurements.

WIZ

Matan Vetzler - matan.vetzler@wiz.io
Last updated: February 2026