# AI-Driven Security Audits vs. Human Audits: A Comprehensive Guide

Security auditing is evolving with AI tools augmenting human expertise. This guide analyzes common mistakes, limitations, and weaknesses of AI-driven security audits across the full workflow – from vulnerability detection and impact analysis to fix suggestions and patch validation – and compares them to traditional human-led audits. Using the **AIxBlock** bug bounty program as a practical context (covering APIs, Solana smart contracts, decentralized compute infrastructure, workflow engines, etc.), we highlight how to effectively leverage AI while avoiding pitfalls. We also outline a checklist for maximizing bug bounty rewards, including CVSS v3.1 severity classification, proper bug/patch submission requirements, proof-of-concept (PoC) quality, and the role of human reviewers.

## AI Tools and Techniques in Security Auditing

Modern security auditors employ a mix of AI-driven tools and classic techniques to uncover vulnerabilities. Common AI tools and techniques include:

* **Large Language Models (LLMs)** – e.g. ChatGPT, Claude, Code Llama – used to explain code, generate hypotheses, and even draft exploit code or patches. LLMs can act as “code assistants,” helping auditors understand complex code flows and identify potential issues[[1]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=1)[[2]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,focus%20on%20nuanced%20attack%20surfaces). Specialized LLM-based auditors (e.g. *Solidity Sentinel* for smart contracts) can scan code and output findings with severity and remediation suggestions[[3]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Our%20testing%20revealed%20impressive%20capabilities,remediation%20steps%20for%20each%20vulnerability)[[4]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Particularly%20noteworthy%20was%20Sentinel%E2%80%99s%20ability,based%20withdrawal%20patterns).
* **Static Analysis Tools** – Traditional static analyzers like CodeQL, Slither (for Solidity), and SonarQube scan source code or binaries without executing them, checking for known vulnerability patterns (buffer overflows, SQL injection, etc.). Many such tools now integrate AI or pattern-learning to reduce noise. For example, Slither and Mythril are used in blockchain audits to catch common flaws[[5]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=The%20space%20that%20used%20to,security%20community%20deserves%20a%20mention), and AI-enhanced static analysis aims to improve accuracy by analyzing code context[[6]](https://arxiv.org/html/2504.18423v1#:~:text=detection%20presents%20unique%20challenges,approach%20focused%20on%20mitigating%20these).
* **Dynamic Analysis & Fuzzing** – AI-powered fuzzers generate large numbers of test inputs to find crashes or logic breaks. Tools like AFL, libFuzzer, Echidna (for Ethereum), or protocol fuzzers enhanced by LLMs[[7]](https://arxiv.org/html/2508.01750v1#:~:text=LLM,for%20testing%20network%20protocol) can explore edge cases humans might miss. AI-based fuzzing can increase code coverage and discover bugs that static analysis would overlook[[8]](https://www.code-intelligence.com/blog/from-dast-to-dawn-why-fuzzing-is-the-better-solution#:~:text=Fuzzing%20also%20helps%20you%20to,or%20manual%20audits).
* **Symbolic Execution** – Tools such as KLEE (for C/C++), Mythril (EVM), or analysis engines for Solana smart contracts attempt to systematically explore program paths using constraint solving. AI techniques are being researched to guide symbolic execution more intelligently, though path explosion remains a challenge. Some AI agents combine static and dynamic methods to bridge these gaps[[9]](https://fuzzinglabs.com/ai-agents-application-testing/#:~:text=AI%20Agents%20For%20Application%20Security,the%20gap%20between%20the%20two).
* **Automated Exploit Generation Agents** – AI agents (like *PentestGPT* or *AutoGPT* variants) can automate parts of penetration testing. They use LLMs to script exploits, perform reconnaissance, and even simulate social engineering. For example, Raxis reports using ChatGPT/Claude to draft exploit PoC code quickly[[2]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,focus%20on%20nuanced%20attack%20surfaces) and employing *Burp Suite AI* for automated web attack simulation[[10]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=concept%20with%20unmatched%20speed.%20,payloads%2C%20helping%20us%20simulate%20real). These tools accelerate routine tasks such as scanning and input crafting.
* **AI-Based Vulnerability Scanners** – Emerging platforms (e.g. LightChaser, SecuredAI, etc.) implement hundreds of vulnerability patterns and use AI/ML to detect issues like misconfigurations or outdated libraries. LightChaser, for instance, runs locally with 1000+ detection patterns to catch “low-hanging fruit” bugs and gas inefficiencies[[11]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=As%20the%20top%20participant%20in,customized%20towards%20your%20project%E2%80%99s%20needs)[[12]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=It%20found%2067%20total%20issues%2C,to%20be%20protected%20better%20and). Such tools can quickly flag style issues or simple security lapses, though they may overwhelm auditors with minor findings.

**Takeaway:** AI tools excel at sifting through vast codebases, automating repetitive tests, and spotting known weaknesses. They act as force multipliers for auditors by providing rapid analysis and even suggesting fixes. However, they are **not silver bullets** – careful integration with human expertise is crucial. In practice, effective audits use AI to cover breadth and humans to ensure depth and validity.

## Comparing AI vs. Human Auditors – Strengths and Weaknesses

Both AI-driven analysis and human-led auditing have unique advantages and failure modes. Understanding where each succeeds or fails helps in creating a balanced approach:

* **Speed and Coverage:** AI can analyze code and systems at machine speed. It processes thousands of lines or numerous endpoints far faster than a human, making it ideal for broad initial vulnerability scanning. For example, an AI-powered scan of AIxBlock’s open-source repository might catch multiple issues in minutes. Humans, on the other hand, are slower and limited by fatigue – but they can **prioritize** better. A human can decide which parts of code or which findings deserve deeper focus, whereas an AI might exhaustively scan everything (including irrelevant parts).
* **Pattern Recognition vs. Creativity:** AI is strong at recognizing known patterns of vulnerabilities. It will reliably flag a classic SQL injection pattern or an unchecked admin privilege function, as these match its training data or rules. Humans might miss some of these “textbook” issues due to oversight. Conversely, human auditors shine in creative thinking and contextual reasoning. They can identify **business logic flaws** or complex exploit chains that don’t match any predefined pattern. As one expert noted, AI often struggles with nuanced, context-specific threats; human testers catch ~85–90% of complex issues like chained exploits, compared to AI’s 50–65% success rate in dynamic environments[[13]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,impact%20risks%20that%20align). Humans can intuitively explore weird edge-case scenarios (e.g. combining minor bugs into a severe exploit) that an AI might never consider.
* **Consistency vs. Judgment:** AI tools are tireless and consistent. They won’t skip steps due to boredom and can repeatedly test every input or configuration. Human auditors may inconsistently cover scope due to time constraints or bias (focusing more on areas they find interesting or have expertise in). However, humans apply **judgment** to distinguish real threats from noise. For instance, an AI might flag dozens of issues (including low-impact or false alarms), whereas a human triager will recognize which of those truly matter for security. Human-led approaches thus tend to have far fewer false positives (often under 10% false positive rate) by focusing on high-impact risks[[14]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=compared%20to%20AI%E2%80%99s%2050,impact%20risks%20that%20align). AI detections, in contrast, can suffer high false positive rates (20–35% in some cases) and still miss contextual nuances[[15]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=vulnerabilities%20as%20an%20initial%20attack,impact%20risks%20that%20align)[[16]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=Human%20Expertise%20Outsmarts%20AI%20in,Pentesting).
* **Contextual Understanding:** Human auditors understand the broader context – system architecture, threat landscape, and business impact. They know, for example, which API endpoints should be public and which should not, or how a Solana program interacts with on-chain governance. AI lacks true understanding of the environment unless explicitly provided. It might flag a function as vulnerable without “knowing” it’s only callable by an internal admin in a safe context. Humans also grasp **severity** better: they can assess how a vulnerability would affect real users or assets, feeding into accurate CVSS scoring. AI might misjudge severity if it’s only looking at code in isolation.
* **Adaptability:** When faced with something novel – a zero-day attack pattern or a custom protocol – humans can adapt their strategy. They research, brainstorm, and test new hypotheses. AI models are limited by their training data and programmed methods; a completely new exploit technique might go unnoticed because “it wasn’t in the playbook.” AI also tends to be **data-constrained** (e.g. an LLM has a knowledge cutoff and fixed context window[[6]](https://arxiv.org/html/2504.18423v1#:~:text=detection%20presents%20unique%20challenges,approach%20focused%20on%20mitigating%20these)), whereas a human can continuously learn and incorporate the latest information. As AIxBlock’s program evolves (new features, new tech like decentralized compute workflows), a human auditor will adjust testing approaches accordingly; a static AI tool might not.
* **Overconfidence and Ethics:** AI, especially generative models, can **sound confident** even when wrong. They don’t have an internal notion of doubt like a human does. A human auditor will typically double-check uncertain findings or mark them as “needs investigation,” whereas an AI might assert “This function is definitely vulnerable” incorrectly. On the flip side, humans can suffer from **confirmation bias** or underestimate a subtle issue that an AI would systematically test. Ethical judgment is another aspect – humans know not to perform dangerous exploits on production or how to avoid violating rules (e.g. refraining from DoS attacks in the bounty scope), whereas an automated tool might need safeguards to not cross those lines[[17]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,prem%2C%20or%20hybrid).

**Summary:** *AI provides speed, breadth, and pattern matching; humans provide creativity, contextual insight, and critical judgment.* The best results come from combining the two – AI handles repetitive tasks and suggests possibilities, and humans validate and dig into the risky areas. As one augmented testing team put it, every AI-derived result is scrutinized by seasoned professionals to eliminate false positives and understand true business impact[[18]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=While%20AI%20dramatically%20amplifies%20our,by%20our%20seasoned%20security%20professionals). AI is a powerful tool, **not a replacement** for human expertise[[19]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=These%20numbers%20underscore%20a%20simple,a%20replacement%20for%20human%20expertise).

## Common Mistakes and Limitations in AI-Driven Audits

When relying on AI for security auditing, several typical errors and weaknesses can arise. Being aware of these common pitfalls is the first step to mitigating them:

* **False Positives from Pattern Matching:** AI-driven scanners often flag issues that *look* like vulnerabilities but are not exploitable in reality. For example, a static analyzer or LLM might see an external call in a smart contract and warn of a reentrancy vulnerability. In practice, there may be a proper guard (e.g. a reentrancy lock or access control) that the AI failed to fully consider[[20]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=of%20the%20codebase%2C%20as%20we,than%20attempting%20direct%20vulnerability%20discovery). These false alarms can waste time as the auditor must manually trace execution paths and validate each case[[21]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Typically%2C%20security%20researchers%20require%20absolute,liability%20rather%20than%20an%20asset). High false-positive rates (20–35% as noted in studies[[16]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=Human%20Expertise%20Outsmarts%20AI%20in,Pentesting)) are a well-known drawback of automated tools – the AI “cries wolf” about issues a human would immediately recognize as non-threats. Over-reliance on AI without verification leads to chasing many ghosts.
* **False Negatives and Missed Edge Cases:** AI tools may *under-report vulnerabilities*, especially unconventional ones. If a flaw doesn’t match known patterns or lies outside the AI’s training distribution, the tool might miss it entirely. For instance, complex multi-step attack paths or subtle logic bugs (like an order-of-operations issue that only matters under specific conditions) can evade automated detection[[22]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=limitations%20that%20underscore%20the%20necessity,to%20validate%20the%20security%20measures)[[23]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Security%20auditors%20play%20several%20irreplaceable,contract%20security%20implications). Large Language Models have a limited context window and knowledge cutoff[[6]](https://arxiv.org/html/2504.18423v1#:~:text=detection%20presents%20unique%20challenges,approach%20focused%20on%20mitigating%20these), meaning they might not connect clues spread across different files or timeframe. Edge cases involving concurrency, race conditions, or cross-component interactions (e.g. a workflow engine and a smart contract interplay) particularly challenge AI reasoning. These gaps mean an AI-based audit might declare “all clear” when a crafty human hacker could still find a hole.
* **Overconfidence and Misleading Explanations:** Generative AI tools often output results with confident language, even when based on shaky reasoning. An AI might produce a detailed impact analysis that sounds convincing but is partly speculative or incorrect. For example, it could “explain” a vulnerability’s impact referencing a CVE or scenario that doesn’t actually apply to the target system – essentially a hallucinated rationale. Researchers note that LLMs can provide plausible but incorrect conclusions or rationales if they lack full context[[24]](https://arxiv.org/html/2504.13474v1#:~:text=context,significantly%20compromised%20by%20contextual%20blindness)[[25]](https://arxiv.org/html/2504.13474v1#:~:text=Consensus%20%231,struggle%20to%20effectively%20identify%20vulnerabilities). This overconfidence can mislead less-experienced auditors into trusting AI findings without sufficient evidence. Always remember: just because the AI provides a fluent explanation doesn’t guarantee it’s accurate.
* **Hallucinated Patches or Fixes:** When asked to suggest fixes, an AI may generate code that looks good but isn’t quite right. Commonly, LLMs *hallucinate* by inventing functions or libraries that don’t exist, or applying a fix that doesn’t actually resolve the root issue. For instance, an AI might recommend a sanitization function with the correct intent but call a non-existent API, or produce a patch that introduces new logic errors. These hallucinations in code are usually obvious once you test or compile the patch (the code fails to run)[[26]](https://simonwillison.net/2025/Mar/2/hallucinations-in-code/#:~:text=A%20surprisingly%20common%20complaint%20I,invent%20methods%20that%20don%E2%80%99t%20exist)[[27]](https://simonwillison.net/2025/Mar/2/hallucinations-in-code/#:~:text=The%20real%20risk%20from%20using,these%20happen%20all%20the%20time). However, more subtle mistakes – e.g. an incomplete fix that addresses the symptom but not all variants of the bug – can slip through if you only read the AI’s patch without rigorous testing. The AI might not understand all side effects in a complex system, so its patch could break functionality elsewhere or degrade performance (imagine an AI suggesting a completely safe but terribly slow input validation for an API, which a human would recognize as impractical).
* **Lack of Contextual Awareness:** AI tools work on the inputs given to them. If they are not supplied with complete context (or cannot process the entire context due to limitations), their outputs can be context-blind. For example, analyzing a piece of code in isolation might make an AI flag a “vulnerability” that is mitigated by surrounding code or configuration not provided in the prompt[[20]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=of%20the%20codebase%2C%20as%20we,than%20attempting%20direct%20vulnerability%20discovery). Similarly, an AI scanning a Solana smart contract might not realize certain calls are protected by off-chain authorities or that the economic context (token values, external protocol integrations) changes the severity. Humans intuit context more easily; AI needs explicit feeding of that information. If you ask an LLM “Is there a flaw in this function?” and omit that the function is behind admin authentication, the AI may give an alarming answer about an issue that real-world attackers couldn’t reach.
* **Volume of Low-Quality Findings:** A related issue is that automated scans can produce an **overwhelming quantity** of findings, many of which are low-priority or purely informational. Tools like LightChaser have been known to output dozens or even hundreds of issues for a project[[12]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=It%20found%2067%20total%20issues%2C,to%20be%20protected%20better%20and) – including minor style inconsistencies or theoretical concerns – which can bury the truly critical findings in noise. An AI-driven audit, if not curated, might dump such a long list of “potential problems” that a solo auditor could be distracted or discouraged. In bug bounty context, submitting a huge list of minor issues is counterproductive (programs typically want quality over quantity). The human skill lies in filtering and highlighting what really matters; AI doesn’t inherently do that prioritization well unless instructed.
* **Difficulty with Novelty and Adaptive Threats:** AI detection is inherently backward-looking – it finds what it was trained to find. Novel vulnerability types or creative attack strategies can fool AI. For example, as AIxBlock involves decentralized infrastructure and AI workflow engines, there may be new classes of logic flaws or AI-specific issues (like prompt injection in AI agents, or leakage of model data) that an off-the-shelf scanner isn’t designed to catch. Humans are better at anticipating how an attacker might exploit something novel. Mainstream reports and experts emphasize that **human creativity and intuition** remain critical for uncovering non-obvious, emergent threats that automated tools overlook[[28]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,even%20against%20advanced%20adversary%20tactics). AI might miss the forest for the trees when a vulnerability arises from a system design quirk rather than a coding mistake.

In summary, AI-driven audits can fall prey to **false positives, missed vulnerabilities, erroneous confidence, and impractical fixes**. These weaknesses underscore the need for human oversight – as Oak Security’s research noted, even advanced AI models required human validation for complex cases, especially involving multiple components or sophisticated access control[[22]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=limitations%20that%20underscore%20the%20necessity,to%20validate%20the%20security%20measures)[[23]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Security%20auditors%20play%20several%20irreplaceable,contract%20security%20implications). Next, we discuss how to mitigate these issues when using AI as a solo security researcher.

## Mitigating AI Audit Issues and Improving Reliability

Working solo with AI tools means you must be the voice of reason and expertise guiding the AI’s output. Here are strategies to mitigate the common issues and boost the reliability of AI-assisted security audits:

* **Always Verify AI Findings Manually:** Treat every AI-generated vulnerability report as a *lead*, not proven fact. You (the human) should **reproduce and validate** each supposed issue. For example, if the AI flags an API endpoint as vulnerable, attempt the exploit yourself (in a safe manner) or craft a quick PoC to see if you can actually break in. If it claims a Solana smart contract has an overflow bug, write a small test or use a Solana sandbox to trigger it. By confirming the behavior, you filter out false positives. As one auditor guideline states: if verifying an AI finding takes longer than finding it manually would have, the tool is a liability[[21]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Typically%2C%20security%20researchers%20require%20absolute,liability%20rather%20than%20an%20asset). So, make the AI work for you by quickly generating hypotheses, but **you must do the legwork of confirmation**.
* **Use Multiple Tools and Cross-Check:** Don’t rely on a single AI tool’s output. Run traditional static analyzers, linters, or other independent scanners and see where results overlap. If both an AI code assistant and a static tool highlight the same suspicious function, it likely deserves attention. Conversely, if an LLM flags something that no other tool or your own code reading supports, be skeptical. You can even use multiple AI models in tandem – for instance, ask ChatGPT and Claude the same security question and compare answers. Research suggests combining models (Mixture-of-Agents) can reduce individual blind spots[[29]](https://arxiv.org/html/2504.18423v1#:~:text=on%20using%20currently%20available%20technologies,source%20or%20closed). Cross-verification extends to fixes too: after applying an AI-suggested patch, run the analyzer again or write a test to ensure the vulnerability is indeed resolved.
* **Provide Adequate Context to AI:** Many AI mistakes come from lack of context, so mitigate this by feeding the AI all relevant information (within reason). When using LLMs for code analysis, include related code (functions, config files, usage examples) so it doesn’t hallucinate something that’s actually defined elsewhere. If you want an impact analysis, tell the AI about the system architecture or user roles if known. For example, *prompt the AI with the threat model*: “This API requires admin token – what’s the impact if bypassed?” This can steer the AI to more realistic conclusions. Tools like Cursor can load an entire codebase context for the AI[[30]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Protocol%20Architecture%20Analysis%3A%20Cursor%20has,passes%20the%20question%20to%20the), which improves accuracy. Always double-check the AI’s assumptions; if it wasn’t told something, it might be assuming defaults incorrectly.
* **Guard Against Hallucinations:** Be on the lookout for AI output that references unfamiliar functions, CVEs, or concepts. When an AI suggests a patch, try to **compile or run the patch** (even in a test environment) to see if it actually works. Hallucinated code will typically fail immediately – which is good, because it forces correction[[27]](https://simonwillison.net/2025/Mar/2/hallucinations-in-code/#:~:text=The%20real%20risk%20from%20using,these%20happen%20all%20the%20time). If the AI cites a CVE or known vulnerability name, verify that reference independently; sometimes AI confuses one issue for another. You can ask the AI to clarify: “Show me where in the code this vulnerability exists” or “Explain how your patch fixes the issue step by step.” If the explanation is vague or circular, it might be bluffing. Use iterative prompting: if the initial answer looks off, give the AI feedback or more info and ask again (many LLMs improve with guided prompts). Ultimately, **never blindly copy-paste AI-generated patches** – review them as critically as you would a junior developer’s code.
* **Focus on High-Impact, Relevant Findings:** To avoid drowning in AI-reported issues, set criteria for what you’ll pursue. Use severity and scope as filters. For instance, instruct your tools (or mentally note) to prioritize findings that could lead to significant damage: auth bypasses, remote code execution, financial theft in the context of AIxBlock. If the AI outputs 50 items including styling suggestions or micro-optimizations, extract the few that are security-relevant. You can configure some AI scanners with rules or ignore lists to cut down noise. When using an LLM, explicitly ask for *“only security-critical issues in the scope”*. By focusing your questions, you push the AI to be more discriminating. Remember that **quality trumps quantity** in bug bounties – one verified critical bug with a clear impact is worth more than ten low-risk warnings.
* **Iteratively Refine and Test Fixes:** When leveraging AI for fix suggestions, use a tight **test loop**. After implementing the AI’s fix (in a fork or test instance), rerun your exploit or input to ensure the vulnerability is indeed closed. Also test that normal functionality still works; sometimes a naive fix can break something else. If issues remain, ask the AI to improve its patch: e.g. “The fix didn’t cover X scenario, how to handle that?” or use another tool to analyze the patched code for any new weaknesses. This iterative approach, akin to having the AI pair-program with you, helps converge on a solid solution. Additionally, compare the AI’s fix with known secure coding practices – e.g. does it align with OWASP recommendations or common library usage? If not, adjust accordingly. You can even prompt the AI for alternative solutions (“What’s another way to fix this?”) and then choose the best one.
* **Maintain a Human-in-the-Loop for Judgment Calls:** Some things AI should not decide for you – severity ratings, final inclusion/exclusion of an issue, and when a report is ready. Use AI to gather data (like CVSS metrics calculation or relevant references), but **you** assign the severity based on the program’s impact guidelines. For example, AIxBlock follows CVSS v3.1 scoring for severity, so calculate the score yourself or verify the AI’s math if it did it. If the AI outputs a CVSS of 7.5 (High) but you realize the attack requires local access or has a complex prerequisite, you might downgrade it. Conversely, if AI underestimates something that you know would be catastrophic, adjust upward with justification. Always align with the bug bounty’s published severity definitions when available. Remember, a human security team will review your submission – they will appreciate accuracy and honesty in how you represent the bug’s impact.
* **Keep Up with AI Improvements and Use Privacy Modes:** The AI field is rapidly evolving. Newer model versions or specialized security models may perform better and reduce some errors (for instance, a fine-tuned model on security code might hallucinate less about patches[[31]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Our%20testing%20revealed%20impressive%20capabilities,remediation%20steps%20for%20each%20vulnerability)[[32]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Particularly%20noteworthy%20was%20Sentinel%E2%80%99s%20ability,based%20withdrawal%20patterns)). Stay updated on the tools – an AI auditing assistant that was state-of-the-art a year ago might have a better successor now. Also, be mindful of confidentiality: if auditing a private codebase or unpublished bug, prefer local or privacy-respecting AI tools[[33]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Perhaps%20the%20most%20pressing%20challenge,and%20legal%20in%20your%20case)[[34]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=To%20avoid%20using%20a%20proprietary,LLM%20and%20its%20security%20protocols). In a bug bounty like AIxBlock’s (which uses a public repo), sharing code with an AI service is less of an issue, but always double-check program rules about disclosure. Use “privacy mode” features or local LLM instances when needed to avoid leaking sensitive info[[35]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=lead%20to%20exposure%20of%20sensitive,and%20legal%20in%20your%20case).
* **Learn from AI and Build Intuition:** Rather than treating the AI as an infallible oracle, use it as a learning aid. Ask *why* it thinks something is a vulnerability, and cross-learn the reasoning. Over time, you’ll start predicting where the AI might err and develop your own quicker judgment. For instance, you might notice “The AI always warns about X, but in this code that’s intended behavior” – that pattern helps you skim results faster. Conversely, if the AI points out obscure issues (like subtle cryptographic weaknesses or deprecated function use), take the opportunity to research those. This improves your personal auditing skills. The end goal is **augmented auditing**: let the AI cover ground and crunch data, while you apply expertise to interpret results. When done right, this combo can outperform either alone, yielding more reliable and comprehensive security assessments[[36]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=AI%20serves%20as%20a%20powerful,is%20thorough%2C%20reliable%2C%20and%20trusted)[[37]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=The%20optimal%20approach%20appears%20to,contracts%20or%20novel%20attack%20vectors).

By conscientiously supervising AI outputs and reinforcing the audit process with human intelligence, a solo auditor can significantly enhance reliability. Think of the AI as a junior assistant – fast and tireless but needing guidance – and yourself as the lead auditor ensuring nothing critical is missed or misreported.

## AI-Based Audit Checklist for Bug Bounty Success

Finally, to maximize bug bounty eligibility and reward potential (especially in programs like AIxBlock’s, with up to **$40k** in total rewards and extra bonuses for patches), use this structured checklist. It covers preparation, execution, and reporting phases of an AI-assisted security audit:

1. **Understand Scope and Rules:**
2. Carefully read the bounty program scope (e.g. AIxBlock’s scope includes its web dashboard, public APIs, smart contracts on Solana, decentralized compute nodes, workflow automation engine, etc.). Identify in-scope assets and out-of-scope areas. Adhere to any testing rules (no DoS, no affecting other users, etc.)[[38]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=A%20Few%20Rules).
3. Note any technology specifics: If Solana programs are in scope, be ready with tools for Rust auditing; if APIs are in scope, prepare to test HTTP endpoints, etc. Align your AI tools to these areas (for instance, use an LLM that understands Rust for smart contracts, or a web scanner for APIs).
4. Ensure compliance with responsible disclosure requirements: do not publicly disclose findings until allowed, and follow the process the program outlines (on AIxBlock, that means using their public GitHub issues template for reports)[[39]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=Submit%20vulnerabilities%20as%20issues%20on,public).
5. **Setup Tools and Environment:**
6. **Fork and Star the Repository (if applicable):** Many programs (like AIxBlock’s) require you to fork their repo and star it[[40]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=1,should%20be%20concise). This not only abides by rules but also gives you a workspace to experiment with code (e.g., applying patches, running tests).
7. Gather your AI arsenal: e.g. an LLM in your IDE (such as Cursor or GitHub Copilot) for code navigation, a static analysis suite (CodeQL, ESLint security rules, Slither for Solidity, etc.), and any necessary fuzzing tools (for web, something like OWASP ZAP; for binaries, AFL; for smart contracts, use a fuzzer or property-based tests).
8. Set up a safe testing environment: use Docker containers, testnet/blockchain simulations, or local debug modes to execute potentially malicious payloads without harming real systems. If testing the AIxBlock platform, create a test account or use provided sandbox environments if any.
9. Optional but useful: prepare a **password manager or dummy credentials** for testing login flows (never use real user data), and have screen recording or screenshot tools ready to capture proof of concepts.
10. **Vulnerability Discovery with AI Assistance:**
11. **Automated Scanning:** Run static analyzers on the codebase to flag obvious vulnerabilities. Let AI help by summarizing the scan results or filtering false positives. For instance, if CodeQL finds 10 possible SQL injections, you can prompt an LLM to assess each finding’s validity quickly. Use AI to generate test payloads for any findings (e.g. ask “give me an example input that would exploit this issue”).
12. **AI Code Review:** Use LLMs to review critical sections of code. Ask questions like “Does this function properly validate user input?” or “Find security weaknesses in this smart contract.” Leverage tools like Cursor’s codebase query to ask about high-level logic across files[[30]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Protocol%20Architecture%20Analysis%3A%20Cursor%20has,passes%20the%20question%20to%20the). The AI might surface things like missing access controls or dangerous uses of APIs.
13. **Dynamic Testing:** With the AI’s help, fuzz inputs and simulate attacks. For web/API, you can have the AI create a list of test cases (SQL injection strings, XSS payloads, etc.) and run them (manually or via a script). For each module (API endpoint, contract method), systematically try erroneous or malicious inputs. Use AI to write small scripts or to automate form submissions, etc.
14. **Monitor for Unusual Behavior:** As you test, keep logs and watch for crashes, error messages, or any response anomalies – these are signs of potential vulnerabilities. AI can assist by parsing log files or core dumps to point out what went wrong.
15. **Iterate**: If an AI hint doesn’t immediately pan out, refine and try again. For example, if the AI suspects a privilege escalation but you couldn’t reproduce it, consider the possibility that more context or a different approach is needed – perhaps ask the AI to outline step-by-step how the attack might work and follow those steps.
16. **Impact Analysis and Severity Classification:**
17. For each discovered issue, determine its impact. Ask: **What can an attacker achieve?** Steal data? Take over accounts? Crash the service? Use the AI to draft an impact summary, then verify every claim. Make sure to include **ease of exploitation** (low complexity, no user interaction?) as the AIxBlock program rewards are based on impact *and* ease of exploit[[41]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=Found%20something%20serious%3F%20We%E2%80%99ll%20make,Here%27s%20the%20payout%20breakdown).
18. Calculate a CVSS v3.1 score for the vulnerability. Identify metrics like Attack Vector (network/adjacent/local), Privileges Required, User Interaction, Confidentiality/Integrity/Availability impact, etc. You can use an online CVSS calculator or even ask an AI to compute it if you feed the characteristics. Double-check the AI’s scoring logic – ensure it aligns with the actual scenario (e.g., if it’s a Solana on-chain bug, Attack Vector is Network and privileges might be None if any user can call the program).
19. Classify the severity (Critical/High/Medium/Low) based on the CVSS base score and the program’s policy. For instance, Critical might be CVSS 9.0–10, High 7.0–8.9, etc. Clearly state this in the report. If the AI wrote an impact description, edit it for accuracy and conciseness – **impact should be crystal clear** to a reviewer (e.g., “This vulnerability allows any user to execute arbitrary code on the server, resulting in full system compromise – **Critical severity (CVSS 10.0)**”).
20. Be truthful and avoid exaggeration. If an issue is theoretical or hard to exploit, don’t oversell it as “high impact.” Remember, human reviewers will determine the final severity and payout, and they value accurate assessments.
21. **Crafting a Clear Report (with AI assistance if needed):**
22. Use the required template (AIxBlock had a “Bug Report” issue template[[42]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=3,via%20comments%20on%20the%20issue)) to structure your report. Typically, include: **Title, Description, Steps to Reproduce, Impact, Supporting Evidence, and Suggested Fix**.
23. **Description:** Explain the vulnerability in plain terms – what is the bug and where does it occur. You can have the AI help summarize the bug or check grammar, but ensure the final wording is yours and precise. For example, “**Vulnerability:** In api/user.js, the updateProfile endpoint does not verify the user’s auth token, allowing an attacker to update any user’s profile data without permission.”
24. **Steps to Reproduce (PoC):** Provide a reliable way to trigger the bug. This can be a series of steps (for a web issue, e.g., an HTTP request with specific parameters) or a short script. Include screenshots or console outputs as proof. If it’s a UI bug or workflow, screenshots are invaluable. For code exploits, show the input and the vulnerable response. AI can assist by formatting this nicely or even generating an example curl command, but **make sure you actually run that PoC** to confirm it works. The bounty guidelines encourage screenshots or video evidence[[43]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=include%3A%20,the%20issue%20or%20pull%20request) – include those (e.g., a screenshot of data you weren’t supposed to access, or a video of exploiting a workflow engine).
25. **Impact:** State the impact clearly and tie it to real risks. E.g., “Impact: This allows theft of other users’ tokens from the Solana program, which could lead to financial loss” – followed by how severe it is (High/Critical) and maybe how many users/assets are affected in a worst case. If applicable, mention if it bypasses security measures or if it could be chained with another bug for greater effect.
26. **Patch/Remediation:** Here’s where you can shine with your provided fix. Describe in words how to fix the issue (“the function should validate the user’s authorization before processing the request”). If you have a code patch, include a link to your fork’s commit or pull request, or a code diff in the report. AIxBlock’s program explicitly offers extra rewards for working patches[[44]](https://x.com/AIxBlock/status/1932652543064354841#:~:text=AIxBlock%20on%20X%3A%20,Extra%20rewards%20for%20patches)[[45]](https://x.com/AIxBlock/status/1945828856553943108#:~:text=AIxBlock%20on%20X%3A%20,making%20a%20real%20impact%2C), so providing a fix can maximize your bounty. Make sure your patch is high quality:
    * It should solve the problem without breaking other functionality.
    * It should follow the project’s coding style and best practices (e.g., use proper input validation routines or known libraries rather than a quick hack).
    * Reference your patch in the report and explain why it fixes the issue (“The patch adds a check require(msg.sender == owner) which ensures only the owner can call this function, preventing unauthorized access.”).
    * If you used AI to help write the patch, test it thoroughly as noted. A *working* patch means maintainers can apply it with confidence.
27. **References (if any):** It adds credibility to link to any relevant references – e.g., CWE entries, OWASP pages, or similar vulnerabilities in other projects. This isn’t always necessary, but if the AI or your research found a related CVE or an article, you could cite it to show this is a recognized type of flaw.
28. **Engage and Iterate with the Project Team:**
29. After submitting, be responsive to any clarifications the maintainers or security team ask. They may request more info or tests. AIxBlock’s process mentions an optional discussion phase where you can collaborate on the fix in a branch[[46]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=5,fix%20in%20your%20PR%20description) – taking part in this can demonstrate your commitment and might increase goodwill (if not the reward directly).
30. If the team points out that something isn’t actually a bug (perhaps you misinterpreted a feature), don’t be discouraged – revisit with your tools, gather more evidence if you still believe it’s a bug, or accept the feedback and apply it to future efforts. Use AI to double-check any new information they provide.
31. Once the vulnerability is validated and patched, follow the disclosure rules. AIxBlock notes that after the fix is live, public disclosure is allowed with approval[[47]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=,is%20permitted%20with%20AIxBlock%E2%80%99s%20approval). Until then, keep details private.
32. **Post-Analysis and Learning:**
33. After the bounty concludes, whether you got a reward or not, analyze what went well and what didn’t. If you missed a critical bug that someone else found, study that bug’s nature – did your AI tools overlook it? Was it an area you skipped? Use this to improve your workflow.
34. If your submission was successful, consider writing a private or public write-up (following disclosure rules) of how AI helped you, noting specific instances (“AI tool X helped find this bug by… but I had to fix a false positive assumption”). This reflection consolidates your learning.
35. Update your toolset and methods as needed. Perhaps you discovered that a certain AI model was more of a hindrance due to noise – you might try another next time, or adjust the prompting strategy.

Using this checklist, an AI-assisted solo auditor can systematically go from reconnaissance to report, ensuring no key step is missed. Actionable highlights include: **always produce a PoC and fix for maximum rewards**, classify severity with CVSS to substantiate your impact, include evidence (screenshots, videos), and follow the program’s submission guidelines to the letter (forks, templates, etc.)[[42]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=3,via%20comments%20on%20the%20issue). Ultimately, remember that human reviewers have the final say in payouts – present your findings in a professional, thorough, and honest manner to earn their trust.

**Conclusion:** AI-driven security audits are a powerful complement to human-led approaches, but they come with their own challenges. By recognizing where AI tools tend to err (false positives, missing context, etc.) and actively countering those weaknesses, researchers can significantly enhance their solo auditing capabilities. The AIxBlock bounty program exemplifies the modern approach: encouraging participants to not only find bugs but also provide fixes and demonstrate impact with high-quality evidence. In such a setting, an auditor who smartly leverages AI for efficiency – while applying human insight for accuracy – stands to reap the biggest rewards[[48]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,Verizon%20DBIR)[[18]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=While%20AI%20dramatically%20amplifies%20our,by%20our%20seasoned%20security%20professionals). Use AI to **work smarter, not carelessly**, and you can uncover deep vulnerabilities, suggest robust patches, and ultimately help secure cutting-edge platforms while earning well-deserved bounties. Happy hunting!

**Sources:**

* AIxBlock Bug Bounty Program – Rules and context[[39]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=Submit%20vulnerabilities%20as%20issues%20on,public)[[42]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=3,via%20comments%20on%20the%20issue)
* Raxis on AI vs Human Penetration Testing – efficacy stats and insights[[13]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,impact%20risks%20that%20align)[[16]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=Human%20Expertise%20Outsmarts%20AI%20in,Pentesting)
* Oak Security on AI-Assisted Audits – AI tool use cases, limitations, need for oversight[[21]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Typically%2C%20security%20researchers%20require%20absolute,liability%20rather%20than%20an%20asset)[[22]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=limitations%20that%20underscore%20the%20necessity,to%20validate%20the%20security%20measures)
* LLM Vulnerability Detection Research – discusses hallucinations and false positive issues[[6]](https://arxiv.org/html/2504.18423v1#:~:text=detection%20presents%20unique%20challenges,approach%20focused%20on%20mitigating%20these)[[49]](https://arxiv.org/html/2504.18423v1#:~:text=Current%20works%20have%20shown%20LLMs,24)
* Simon Willison’s notes on LLM code hallucinations – importance of testing AI-generated code[[26]](https://simonwillison.net/2025/Mar/2/hallucinations-in-code/#:~:text=A%20surprisingly%20common%20complaint%20I,invent%20methods%20that%20don%E2%80%99t%20exist)[[27]](https://simonwillison.net/2025/Mar/2/hallucinations-in-code/#:~:text=The%20real%20risk%20from%20using,these%20happen%20all%20the%20time)
* LightChaser audit output example – illustrating volume of findings and need for human judgment[[12]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=It%20found%2067%20total%20issues%2C,to%20be%20protected%20better%20and)
* AI tools in augmented testing – examples of usage (PentestGPT, Burp Suite AI, etc.)[[2]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,focus%20on%20nuanced%20attack%20surfaces)
* AIxBlock Blog – bounty incentive details and submission process[[41]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=Found%20something%20serious%3F%20We%E2%80%99ll%20make,Here%27s%20the%20payout%20breakdown)[[43]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=include%3A%20,the%20issue%20or%20pull%20request)

[[1]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=1) [[3]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Our%20testing%20revealed%20impressive%20capabilities,remediation%20steps%20for%20each%20vulnerability) [[4]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Particularly%20noteworthy%20was%20Sentinel%E2%80%99s%20ability,based%20withdrawal%20patterns) [[5]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=The%20space%20that%20used%20to,security%20community%20deserves%20a%20mention) [[11]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=As%20the%20top%20participant%20in,customized%20towards%20your%20project%E2%80%99s%20needs) [[12]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=It%20found%2067%20total%20issues%2C,to%20be%20protected%20better%20and) [[20]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=of%20the%20codebase%2C%20as%20we,than%20attempting%20direct%20vulnerability%20discovery) [[21]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Typically%2C%20security%20researchers%20require%20absolute,liability%20rather%20than%20an%20asset) [[22]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=limitations%20that%20underscore%20the%20necessity,to%20validate%20the%20security%20measures) [[23]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Security%20auditors%20play%20several%20irreplaceable,contract%20security%20implications) [[30]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Protocol%20Architecture%20Analysis%3A%20Cursor%20has,passes%20the%20question%20to%20the) [[31]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Our%20testing%20revealed%20impressive%20capabilities,remediation%20steps%20for%20each%20vulnerability) [[32]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Particularly%20noteworthy%20was%20Sentinel%E2%80%99s%20ability,based%20withdrawal%20patterns) [[33]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=Perhaps%20the%20most%20pressing%20challenge,and%20legal%20in%20your%20case) [[34]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=To%20avoid%20using%20a%20proprietary,LLM%20and%20its%20security%20protocols) [[35]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=lead%20to%20exposure%20of%20sensitive,and%20legal%20in%20your%20case) [[37]](https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be#:~:text=The%20optimal%20approach%20appears%20to,contracts%20or%20novel%20attack%20vectors) AI-Assisted Security Audits. A Practical Guide with Real-World… | by Eduard Kotysh | Oak Security | Medium

<https://medium.com/oak-security/ai-assisted-security-audits-0bd76608e3be>

[[2]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,focus%20on%20nuanced%20attack%20surfaces) [[10]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=concept%20with%20unmatched%20speed.%20,payloads%2C%20helping%20us%20simulate%20real) [[13]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,impact%20risks%20that%20align) [[14]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=compared%20to%20AI%E2%80%99s%2050,impact%20risks%20that%20align) [[15]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=vulnerabilities%20as%20an%20initial%20attack,impact%20risks%20that%20align) [[16]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=Human%20Expertise%20Outsmarts%20AI%20in,Pentesting) [[17]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,prem%2C%20or%20hybrid) [[18]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=While%20AI%20dramatically%20amplifies%20our,by%20our%20seasoned%20security%20professionals) [[19]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=These%20numbers%20underscore%20a%20simple,a%20replacement%20for%20human%20expertise) [[28]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,even%20against%20advanced%20adversary%20tactics) [[36]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=AI%20serves%20as%20a%20powerful,is%20thorough%2C%20reliable%2C%20and%20trusted) [[48]](https://raxis.com/pentest/ai-vs-human-pentest/#:~:text=,Verizon%20DBIR) AI vs. Human Penetration Testing | Raxis

<https://raxis.com/pentest/ai-vs-human-pentest/>

[[6]](https://arxiv.org/html/2504.18423v1#:~:text=detection%20presents%20unique%20challenges,approach%20focused%20on%20mitigating%20these) [[29]](https://arxiv.org/html/2504.18423v1#:~:text=on%20using%20currently%20available%20technologies,source%20or%20closed) [[49]](https://arxiv.org/html/2504.18423v1#:~:text=Current%20works%20have%20shown%20LLMs,24) LLMpatronous: Harnessing the Power of LLMs For Vulnerability Detection

<https://arxiv.org/html/2504.18423v1>

[[7]](https://arxiv.org/html/2508.01750v1#:~:text=LLM,for%20testing%20network%20protocol) LLM-Assisted Model-Based Fuzzing of Protocol Implementations

<https://arxiv.org/html/2508.01750v1>

[[8]](https://www.code-intelligence.com/blog/from-dast-to-dawn-why-fuzzing-is-the-better-solution#:~:text=Fuzzing%20also%20helps%20you%20to,or%20manual%20audits) From DAST to dawn: why fuzzing is better solution - Code Intelligence

<https://www.code-intelligence.com/blog/from-dast-to-dawn-why-fuzzing-is-the-better-solution>

[[9]](https://fuzzinglabs.com/ai-agents-application-testing/#:~:text=AI%20Agents%20For%20Application%20Security,the%20gap%20between%20the%20two) AI Agents For Application Security Testing - Fuzzing Labs

<https://fuzzinglabs.com/ai-agents-application-testing/>

[[24]](https://arxiv.org/html/2504.13474v1#:~:text=context,significantly%20compromised%20by%20contextual%20blindness) [[25]](https://arxiv.org/html/2504.13474v1#:~:text=Consensus%20%231,struggle%20to%20effectively%20identify%20vulnerabilities) Everything You Wanted to Know About LLM-based Vulnerability Detection But Were Afraid to Ask

<https://arxiv.org/html/2504.13474v1>

[[26]](https://simonwillison.net/2025/Mar/2/hallucinations-in-code/#:~:text=A%20surprisingly%20common%20complaint%20I,invent%20methods%20that%20don%E2%80%99t%20exist) [[27]](https://simonwillison.net/2025/Mar/2/hallucinations-in-code/#:~:text=The%20real%20risk%20from%20using,these%20happen%20all%20the%20time) Hallucinations in code are the least dangerous form of LLM mistakes

<https://simonwillison.net/2025/Mar/2/hallucinations-in-code/>

[[38]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=A%20Few%20Rules) [[39]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=Submit%20vulnerabilities%20as%20issues%20on,public) [[40]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=1,should%20be%20concise) [[41]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=Found%20something%20serious%3F%20We%E2%80%99ll%20make,Here%27s%20the%20payout%20breakdown) [[42]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=3,via%20comments%20on%20the%20issue) [[43]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=include%3A%20,the%20issue%20or%20pull%20request) [[46]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=5,fix%20in%20your%20PR%20description) [[47]](https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000#:~:text=,is%20permitted%20with%20AIxBlock%E2%80%99s%20approval) Think You’ve Got What It Takes? Prove It and Win $40,000! | Decentralized AI Development & Workflow Automation Platform

<https://aixblock.io/blog/think-youve-got-what-it-takes-prove-it-and-win-40000>

[[44]](https://x.com/AIxBlock/status/1932652543064354841#:~:text=AIxBlock%20on%20X%3A%20,Extra%20rewards%20for%20patches) AIxBlock on X: "🛡️ AIxBlock just launched a $40,000 Bug Bounty ...

<https://x.com/AIxBlock/status/1932652543064354841>

[[45]](https://x.com/AIxBlock/status/1945828856553943108#:~:text=AIxBlock%20on%20X%3A%20,making%20a%20real%20impact%2C) AIxBlock on X: " Reminder: Our $40,000 Bug Bounty Program is still ...

<https://x.com/AIxBlock/status/1945828856553943108>