Absolutely! I understand. You want me to create a comprehensive plan that you can take to a new workspace to properly execute the AIxBlock bug bounty, while keeping our current development work separate and intact.  
📋 Complete Bug Bounty Plan for New Workspace  
🚨 Where We Went Wrong  
Wrong Target: We analyzed our own mock/development code instead of AIxBlock's official production codebase  
Wrong Scope: We focused on functionality bugs (React imports, ConnectKit config) instead of security vulnerabilities  
Wrong Classification: We submitted stability issues as "security vulnerabilities"  
Wrong Repository: We worked in a development environment rather than their official bug bounty target  
✅ Correct Approach - Step-by-Step Plan  
PHASE 1: Environment Setup (New Workspace)  
1.1 Repository Setup  
  
# In your NEW workspace:  
mkdir aixblock-security-audit  
cd aixblock-security-audit  
  
# Clone the OFFICIAL AIxBlock repository  
git clone https://github.com/AIxBlock-2023/awesome-ai-dev-platform-opensource.git  
cd awesome-ai-dev-platform-opensource  
  
# Star and fork the repository (mandatory per their rules)  
gh repo fork --clone=false  
gh api user/starred/AIxBlock-2023/awesome-ai-dev-platform-opensource --method PUT  
  
1.2 Tools Setup  
  
# Install security testing tools npm install -g @eslint/eslintrc eslint-plugin-security pip install bandit safety semgrep npm install -g retire snyk  
  
PHASE 2: Security Audit Methodology  
2.1 Target Analysis (Based on Their Scope)  
Primary Targets (from their bounty scope):  
app.aixblock.io - Web Application (High value)  
api.aixblock.io - API Endpoints (Critical value)  
smartcontracts.aixblock.io - Solana Smart Contracts (Critical value)  
workflow.aixblock.io - Workflow Engine (Critical value)  
\*.aixblock.io - All subdomains (Medium value)  
2.2 Vulnerability Categories to Focus On  
Critical ($750 + 1,500 USDC):  
Remote Code Execution (RCE)  
Smart Contract logic flaws causing asset loss  
Data leakage of AI models/workflow data  
Unauthorized workflow execution  
High ($450 + 1,000 USDC):  
Server-Side Request Forgery (SSRF)  
Authentication Bypass  
Unauthorized access to compute/workflow triggers  
Medium ($200 + 500 USDC):  
Cross-Site Scripting (XSS) affecting workflow actions  
CSRF affecting workflow actions  
Webhook misconfiguration  
PHASE 3: Systematic Security Testing  
3.1 Static Code Analysis  
  
# Scan for security vulnerabilities in the codebase  
semgrep --config=auto .  
bandit -r . -f json -o security-report.json  
npm audit  
retire --path .  
  
3.2 Authentication & Authorization Testing  
Focus Areas:  
JWT token handling and validation  
Session management  
Role-based access controls  
API authentication mechanisms  
OAuth/SSO implementations  
Test Cases:  
Token manipulation/forgery  
Session fixation  
Privilege escalation  
Authentication bypass  
Insecure direct object references  
3.3 API Security Testing  
Target: api.aixblock.io/api/\*  
SQL injection in API parameters  
NoSQL injection (if using MongoDB/similar)  
Command injection in workflow parameters  
Path traversal in file operations  
Rate limiting bypass  
API versioning vulnerabilities  
3.4 Smart Contract Audit  
Target: Solana contracts on smartcontracts.aixblock.io  
Reentrancy attacks  
Integer overflow/underflow  
Access control flaws  
Logic bugs in payment/compute allocation  
Unauthorized token transfers  
3.5 Web Application Security  
Target: app.aixblock.io  
XSS (stored, reflected, DOM-based)  
CSRF in critical operations  
Clickjacking  
Open redirects  
File upload vulnerabilities  
Client-side security controls bypass  
PHASE 4: Workflow-Specific Security  
4.1 Workflow Engine Vulnerabilities  
Target: workflow.aixblock.io  
Code injection in workflow definitions  
Unsafe deserialization  
Server-side template injection  
File inclusion vulnerabilities  
Sandbox escape  
4.2 AI/ML Specific Attacks  
Model poisoning  
Training data extraction  
Adversarial inputs  
Model inversion attacks  
Prompt injection (if using LLMs)  
PHASE 5: Proof of Concept Development  
5.1 For Each Vulnerability Found:  
Document the vulnerability with clear steps to reproduce  
Assess the impact using CVSS v3.1 scoring  
Create a working exploit (PoC)  
Develop a fix/patch  
Test the fix to ensure it works  
5.2 Documentation Template:  
  
# Vulnerability Title  
\*\*Severity:\*\* Critical/High/Medium/Low  
\*\*CVSS Score:\*\* X.X  
\*\*Asset:\*\* app.aixblock.io / api.aixblock.io / etc.  
  
## Description  
[Clear description of the vulnerability]  
  
## Impact  
[Business impact and potential damage]  
  
## Steps to Reproduce  
1. [Step 1]  
2. [Step 2]  
3. [Step 3]  
  
## Proof of Concept  
[Working exploit code/screenshots]  
  
## Recommended Fix  
[Specific code changes needed]  
  
## References  
[OWASP, CVE references if applicable]  
  
HASE 6: Submission Process  
6.1 Pre-Submission Checklist:  
[ ] Vulnerability affects their official codebase (not mock/dev code)  
[ ] Clear security impact (not just functionality bugs)  
[ ] Working proof of concept  
[ ] Proposed fix/patch  
[ ] Proper CVSS classification  
[ ] Screenshots/video evidence  
6.2 Submission Steps:  
Create issue using their bug report template  
Include all required information per their guidelines  
Create a branch for your proposed fix  
Submit pull request with the security patch  
Engage professionally with their security team  
PHASE 7: Tools & Resources  
7.1 Security Testing Tools:  
Burp Suite - Web application security testing  
OWASP ZAP - Free web app scanner  
Nuclei - Vulnerability scanner  
Subfinder - Subdomain enumeration  
Nmap - Network scanning  
Metasploit - Exploitation framework  
7.2 Smart Contract Tools:  
Anchor - Solana development framework  
Solana CLI - Command line tools  
Mythril - Smart contract security analyzer  
7.3 Resources:  
OWASP Top 10 - Web application vulnerabilities  
OWASP API Security Top 10 - API vulnerabilities  
Solana Security Best Practices  
CVSS v3.1 Calculator - For severity scoring  
🎯 Key Success Factors  
Focus on REAL security vulnerabilities (not functionality bugs)  
Target their PRODUCTION systems (not development/mock code)  
Provide working exploits and fixes  
Follow their submission guidelines exactly  
Engage professionally with their security team  
Be patient - security reviews take time  
📝 Final Notes  
Keep this current workspace with our development/testing work separate  
Use the new workspace exclusively for the legitimate security audit  
Document everything thoroughly  
Focus on high-impact vulnerabilities first  
Remember: Quality over quantity - one critical vulnerability is better than ten low-impact issues  
This plan gives you a complete roadmap to execute a proper security audit in your new workspace. The key is targeting their actual production systems and finding real security vulnerabilities, not the development/mock code we created for testing.  
  
Here is the aixblock bug bounty description (plus images) -   
  
Overview  
AIxBlock combines decentralized compute, open‑source models, data engines and human validators into a low‑code environment for end‑to‑end AI and workflow automation. Our goal is to identify and remediate vulnerabilities quickly and transparently while crediting and rewarding the researchers who make that possible.  
  
All vulnerability discussion and fixes are public (issues & PRs).  
  
Rewards scale with impact and fix quality – submit the bug and a working patch to maximize your payout.  
  
We follow the [CVSS v3.1] qualitative scale for severity (see footnote 1).  
  
Scope  
  
Out of Scope  
Third‑party services we don’t control (e.g. Solana L1, Hugging Face, Roboflow).  
  
DoS / DDoS or spam/flood tests.  
  
UI bugs with no security impact.  
  
Proprietary/private models or data not present in the public repo.  
  
Rules of Engagement  
Be lawful & respectful – no social engineering, physical attacks, or privacy violations.  
  
Only test with accounts you own or explicit permission.  
  
No public disclosure until the fix is merged (see timeline below).  
  
Chain or duplicate vulnerabilities = one bounty.  
  
First valid report wins if duplicates occur.  
  
Eligibility  
Anyone can participate except:  
  
Current AIxBlock employees or contractors.  
  
Reporting Process  
Submit vulnerabilities as issues on the public repository at https://github.com/AIxBlock-2023/aixblock-ai-dev-platform-public. Follow these essential steps:  
  
Star the Repository (mandatory): Stay updated and show your engagement.  
  
Fork the Repository (mandatory): Fork to contribute, keep track of your changes and use as a proof to claim tokens later as well as to receive long term revenue sharing in the future.  
  
Submit Report:  
  
Create an issue using the "Bug Report" template (if available), and include:  
  
🔍 Vulnerability description  
  
🧠 Impact assessment (should be concise)  
  
📸 Screenshots or video evidence  
  
Discussion (optional, but encouraged):  
  
Create a dedicated branch (e.g., bugfix/issue-123) to collaborate on your proposed fix.  
  
Engage with the AIxBlock team and the community via comments on the issue or pull request.  
  
Fix bug/propose solutions and pull PR request :  
  
Submit a pull request (PR) to the discussion branch.  
  
Reference the original issue in your PR.  
  
Include description of the fix in your PR description.  
  
AIxBlock Responds:  
  
The AIxBlock Security Team will acknowledge your submission within 48 hours.  
  
AIxBlock Validates:  
  
Vulnerabilities are validated within 7 business days.  
  
Severity and reward are confirmed after validation.  
  
Disclosure:  
  
Once the fix is live and merged, public disclosure is permitted with AIxBlock’s approval.  
  
Severity and Rewards  
  
\*Token: payouts redeemable on or one day after TGE.  
  
Bonus: Additional for reports with detailed PoCs or vulnerabilities found in new features (e.g., automation workflows, MCP integration, decentralized compute).  
  
No Reward: Duplicate reports, out-of-scope issues, or vulnerabilities with no security impact.  
  
Payments: Cash Rewards can be paid via either bank transfer in fiat currency or in stablecoins (USDC) as cryptocurrency payment, distributed at the end of the bounty campaign once the total cash rewards pool reaches 10,000 USD, will be announced publicly on all of our channels. Token rewards shall be distributed on our TGE date or 1 day after our TGE date. Please follow us to make sure you don't miss it.  
  
Total pool: $10 000 cash + $30 000  worth of AXB tokens.  
Response Targets  
  
  
Future Benefits  
Additional opportunities to claim tokens on our TGE date and receive long-term revenue sharing.  
  
Contact  
Discord: Join Us  
  
Twitter: Follow Us  
  
Telegram: Join the Discussion  
  
LinkedIn: Follow Us  
  
YouTube: Watch Our Channel  
  
Website: https://aixblock.io  
  
Platform: https://app.aixblock.io  
  
Huggingface: https://huggingface.co/AIxBlock.  
  
Footnotes  
CVSS v3.1 Severity Rating Scale – FIRST.org.  
  
Lets get this bounty going and win it  
  
please review the below and use all tools to confirm current analysis and add additional penetration and security flaw detection -   
  
Open-Source Security Tools for AIxBlock Systems  
  
AIxBlock’s bug bounty scope spans multiple domains – from traditional web apps and APIs to Solana smart contracts, decentralized compute nodes, data engines, webhooks, and the “MCP” integration layer. Below is a comprehensive, categorized list of free/open-source tools (and example commands) used to identify vulnerabilities in each domain and help patch/remediate them. Tools are grouped by domain type and further organized by vulnerability category for clarity.  
  
Web Applications  
  
Web apps face a wide range of vulnerabilities (RCE, XSS, SSRF, CSRF, etc.). The tools below help uncover and fix such issues:  
  
Remote Code Execution (RCE) & Injection  
  
OWASP ZAP – A full-featured web app scanner/proxy that can find injections (SQL, command) and more  
hackerone.com  
hackerone.com  
. Usage: Launch the ZAP GUI or run headless (e.g. zap.sh -daemon -port 8090) and use the built-in Active Scan to probe for RCE and SQLi. ZAP’s active scan will attempt payloads for OS command injection, SQL injection, etc., flagging any code execution flaws  
hackerone.com  
. Docs:   
hackerone.com  
hackerone.com  
  
Wapiti – A black-box web vulnerability scanner that fuzzes parameters to detect SQLi, XSS, file inclusion, command execution (e.g. via eval()), XXE, SSRF and more  
kali.org  
kali.org  
. Usage: Run wapiti -u https://target/site to crawl and attack; include -m exec to focus on command exec tests. Wapiti injects payloads and can catch OS command injection (e.g. via system calls) and even Shellshock exploits  
kali.org  
. It’s CLI-driven and outputs a report of vulnerabilities found.  
  
sqlmap – An open-source SQL injection tool that automates finding and exploiting SQLi to achieve DB takeover and even underlying OS code execution  
sqlmap.org  
sqlmap.org  
. Usage: sqlmap -u "http://site/page.php?id=1" --dbs will detect an SQLi on the id parameter and list databases  
kali.org  
kali.org  
. Once a SQL injection is found, sqlmap can use out-of-band techniques to execute OS commands on the DB server  
sqlmap.org  
sqlmap.org  
(if, for example, xp\_cmdshell or similar is available). This helps confirm RCE via the database. Docs:   
sqlmap.org  
kali.org  
  
Commix – A command injection exploit tool that tests parameters for OS command injection (useful for RCE). Usage: commix -u "http://site/page?param=val" tries various payloads to drop into system shells. (Commix is fully open-source, specialized for OS command injections.)  
  
Nuclei – A template-driven scanner that can test for known RCE exploits and config issues. By using community templates (YAML files), Nuclei can quickly detect unpatched RCE vulnerabilities in web software. Usage: nuclei -u https://target -t rce.yaml (with appropriate templates for CVE checks). It’s effective for scanning known CVEs in web apps at scale.  
  
Cross-Site Scripting (XSS)  
  
XSStrike – An advanced XSS detection suite with intelligent fuzzing  
github.com  
. It parses responses and crafts payloads that are contextually likely to execute  
github.com  
. Usage: python xsstrike.py -u "http://site/search?q=test" will crawl and test parameters for reflected, stored, and DOM XSS. XSStrike’s payload generator and fast crawler help find tricky XSS that other scanners miss (e.g. complex tag injections)  
github.com  
github.com  
. Source:   
github.com  
  
OWASP ZAP (again) – ZAP’s active scanner includes XSS checks (inserting common <script> tags, event handlers, etc.). It flags XSS if payloads are reflected unencoded  
hackerone.com  
. Usage: Use Passive Scan mode while browsing to catch reflected XSS in pages, then Active Scan specific parameters to confirm. ZAP’s fuzzer can also be used with a list of XSS payloads.  
  
Wapiti – Wapiti explicitly checks for reflected and permanent XSS  
kali.org  
kali.org  
. It injects XSS strings and observes if they appear unsanitized. Usage: wapiti -u https://site -m xss to run only the XSS module. It will report any reflected payloads (like <script>alert(1)</script>) that get executed or appear in responses.  
  
Dalfox – A fast parameter analyzer and XSS scanner (popular in bug bounty circles). Usage: dalfox url https://target/page?param=value hunts for XSS by param fuzzing. It’s known for detecting tricky cases like DOM-based XSS by analyzing scripts.  
  
Server-Side Request Forgery (SSRF)  
  
SSRFmap – Automatic SSRF fuzzer/exploitation tool  
cybersectools.com  
. It takes a crafted request (e.g. exported from Burp) and injects payloads to trigger SSRF, attempting to reach internal services  
github.com  
. Usage: Save a vulnerable request (with a URL parameter) as req.txt and run ssrfmap -r req.txt -p url (replace “url” with the param name). It will try common SSRF payloads (http://127.0.0.1:22, file:///etc/passwd, etc.) and report any responses  
cybersectools.com  
. This helps find backend network access or file reads via SSRF. Docs:   
cybersectools.com  
  
RequestBin/Webhook Site – While not a scanner, these free tools provide a public endpoint to catch SSRF. You can direct potential SSRF payloads to https://<your-bin>.requestbin.net and see if the web app makes an outgoing request. Usage: Insert http://your-bin in suspected SSRF parameters (like image URLs) and monitor the bin for hits – if you see one, SSRF is confirmed.  
  
OWASP ZAP – ZAP can detect some SSRF indicators (e.g. query responses suggesting internal IP leakage). It’s not fully automatic for SSRF, but ZAP’s fuzzing can be used to insert internal addresses in parameters and watch responses.  
  
Authentication Bypass / Unauthorized Access  
  
Autoswagger – An open-source tool to scan OpenAPI/Swagger-defined APIs for broken authentication and authorization controls  
helpnetsecurity.com  
. It auto-discovers API endpoints from docs and sends requests with and without auth to find endpoints that improperly accept requests  
helpnetsecurity.com  
helpnetsecurity.com  
. Usage: Provide an OpenAPI spec or domain to Autoswagger; it will enumerate all operations. For each, it sends requests without credentials – if a 200 OK is returned where a 401/403 is expected, it flags a missing auth check  
helpnetsecurity.com  
. It also checks for endpoints that reveal data to unauthorized users. This is very useful in finding IDORs or endpoints with no auth. Source:   
helpnetsecurity.com  
  
OWASP ZAP (Forced Browse & Authenticated Scan) – ZAP’s Forced Browse (using wordlists) can find hidden files or admin panels not protected by auth. Usage: Use the “Forced Browse” add-on with a common files list; any sensitive page discovered (like /admin.jsp) can indicate a potential auth bypass if not properly locked down. ZAP can also be scripted with context-based scanning to test privilege escalation (login as low-priv user and attempt admin-only URLs).  
  
JWT Tool – If the app uses JSON Web Tokens, open-source tools like jwt-cli or JWT Crack can test token strength. Usage: jwt decode <token> to inspect algorithm (alg). If “none” or a weak HMAC is allowed, that’s an auth bypass vector. Example: jwt brute -t <token> -w common-secrets.txt attempts to find the signing secret (common flaw if secret is weak). This helps expose poorly secured token-based auth.  
  
Hydra (Credential Brute-force) – For testing authentication hardening, Hydra is open-source and can brute-force login forms (online). Usage: hydra -l user -P passwords.txt https://site/login.php http-post-form "user=^USER^&pass=^PASS^:Invalid login" to identify weak credentials (though use ethically and within bounty rules). This can uncover a trivial auth bypass via default or weak passwords.  
  
Cross-Site Request Forgery (CSRF)  
  
OWASP ZAP – ZAP can passively alert on forms missing anti-CSRF tokens. Its passive scanner flags any HTML forms that lack CSRF prevention measures (like a hidden token or same-site cookies). Look for “CSRF Token Missing” alerts in ZAP’s report – these indicate potential CSRF vulnerabilities.  
  
Burp Suite (Community Edition) – While not fully open source, the free edition’s CSRF PoC generator is useful. You can craft a CSRF proof-of-concept HTML form for any state-changing request and test if it executes (e.g., change user email without token). This effectively checks if CSRF protection is in place.  
  
Manual: Often CSRF is best verified manually: for instance, intercept a POST request, remove the CSRF token or session cookie, and replay it. If the action still succeeds, the endpoint is vulnerable. Tools like cURL can automate this (e.g. send the request without token and see if 200 OK).  
  
Sensitive Data Exposure & Info Leakage  
  
Nikto – A classic web server scanner that checks for thousands of known vulnerable files, config issues, and default credentials  
en.wikipedia.org  
en.wikipedia.org  
. It can find backup files, config dumps, or version disclosures. Usage: nikto -h <website> performs checks for sensitive files (like admin.conf, phpinfo.php), outdated server software, default admin pages, etc.  
en.wikipedia.org  
en.wikipedia.org  
. For example, Nikto might alert on an open .git/ directory or a publicly accessible config file (which could leak credentials)  
en.wikipedia.org  
en.wikipedia.org  
. Docs:   
en.wikipedia.org  
en.wikipedia.org  
  
Dirsearch / FFUF – These directory brute-forcers find hidden files or folders that could contain sensitive data (like /backup/, /config/dev.yaml). Usage: ffuf -u https://site/FUZZ -w common.txt – any discovered endpoints can then be examined for secrets. For instance, finding /env or /config.php~ could reveal passwords or keys (data leakage issues).  
  
TruffleHog – A powerful secrets scanner that searches code repos and files for API keys, passwords, and tokens  
trufflesecurity.com  
. While typically used on code, it can be run on a deployed site’s public repository if accessible. Usage: trufflehog git https://github.com/org/app.git scans the git history for high-entropy strings and known secret patterns (AWS keys, etc.)  
trufflesecurity.com  
qualt.io  
. This is useful if, say, an open .git folder was found by Dirsearch – you could extract it and run TruffleHog to see if any committed secrets are present. Docs:   
trufflesecurity.com  
qualt.io  
  
Gitleaks – Similar to TruffleHog, an open-source SAST tool for hardcoded secrets in git repos. It’s fast and can be integrated into CI. Usage: gitleaks detect --source=. --redact (run in a repository) will flag any API keys, tokens, or passwords in the code history. Gitleaks helps prevent accidental data leakage of secrets by scanning before code deploys  
github.com  
. Docs:   
github.com  
  
TestSSL.sh – A CLI tool to check a web server’s SSL/TLS configuration for known weaknesses  
kali.org  
. While not a “vulnerability” in the app per se, weak encryption is a configuration issue that can lead to data exposure. Usage: testssl.sh https://your-site will report on supported ciphers, protocol versions, and known TLS flaws (BEAST, POODLE, Heartbleed, etc.)  
kali.org  
. This helps ensure data in transit is properly protected. Source:   
kali.org  
  
Minor Config Issues & Hardening  
  
Lynis – A Unix/Linux security auditing tool that checks system and software configuration for weaknesses (missing patches, bad permissions, etc.). Though typically run on servers, for web apps it can highlight OS-level issues that allow escalation. Usage: Run lynis audit system on the web server; it will output warnings and suggestions (e.g. outdated Apache version, world-writable files) to harden the setup.  
  
CIS-CAT Lite / OpenSCAP – These can be used to assess the server against benchmarks (CIS). They’ll flag misconfigurations in the web stack (like directory listing enabled, debug mode on, etc.). Ensuring these are fixed reduces the chance of low-hanging configuration flaws.  
  
Package Managers (for patching) – To remediate issues, use system package tools and dependency checks:  
  
For OS: apt-get upgrade, etc., to apply patches once a scan (like Lynis or OpenVAS) shows missing updates.  
  
For web app libraries: OWASP Dependency-Check finds known vulnerable components in the app’s dependencies  
github.com  
. Usage: dependency-check --project WebApp --scan ./app generates an HTML report of libraries with CVEs  
github.com  
. This helps identify outdated frameworks (e.g. a vulnerable Struts version leading to RCE). Docs:   
github.com  
  
Language-specific: npm audit (Node.js), pip-audit or Safety (Python), gradle dependencyCheck (Java with OWASP plugin) – all flag libraries that need patching. These tools assist in patching by suggesting fixed versions.  
  
APIs  
  
APIs (REST/GraphQL) often suffer from issues like authorization flaws, injections, and data exposure. Open-source tools effective in bug bounties include:  
  
Injection & Remote Vulnerabilities (API)  
  
OWASP ZAP (API Scan) – ZAP can import an API’s OpenAPI/Swagger definition and actively scan all endpoints  
hackerone.com  
. Usage: Use the ZAP API scanner add-on or zap-api-scan.py script with the API’s JSON definition. ZAP will fuzz query/body parameters for SQLi, XSS (if responses are rendered), and command injection. For example, it can detect SQL injection in an API parameter (just like for web forms) and issues like XML External Entity (XXE) if the API parses XML input.  
  
Postman / Insomnia – These are API clients (Insomnia is open-source) useful for manual testing. While not automated scanners, they let you systematically send requests. Combined with a fuzzing extension or by importing wordlists, you can test for SQLi (e.g. entering ' OR '1'='1 in parameters and checking responses) and SSRF in any URL parameters. Manual testing with these tools is key for APIs.  
  
sqlmap – Works on APIs too. If an API endpoint is vulnerable (say a JSON body with an injectable field), you can feed sqlmap: sqlmap -u https://api/site/endpoint -X POST --data='{"id":1}' -p id. Sqlmap will automate SQL injection tests on the JSON parameter and even exploit it if possible  
kali.org  
kali.org  
.  
  
NoSQLMap – If the API uses NoSQL (MongoDB, etc.), NoSQLMap (open-source) can test NoSQL injection (for example, sending {"$ne": null} in JSON to bypass auth). It’s a more specialized tool to check for NoSQL injection flaws that might lead to data leakage or RCE in Mongo/NoSQL backends.  
  
CRLFuzz – Some APIs might be vulnerable to HTTP header injections (CRLF). Tools like crlfuzz can test endpoints by injecting %0d%0a and seeing if header injection or response splitting occurs.  
  
Authentication & Authorization (Broken Auth)  
  
Autoswagger – (Mentioned above under Web Auth) is extremely useful for APIs. It scans OpenAPI specs for endpoints that don’t enforce auth  
helpnetsecurity.com  
. For a bounty hunter, running Autoswagger on the target’s API spec quickly highlights any endpoint that responds without proper auth (e.g., an admin-only endpoint that’s publicly accessible)  
helpnetsecurity.com  
. It automates what would otherwise be a manual, tedious process. Docs:   
helpnetsecurity.com  
helpnetsecurity.com  
  
JWT Inspector – Many APIs use JWT for stateless auth. Open-source libraries or CLI tools (e.g. jwt-cli) let you decode and check JWTs. Look for algorithm flaws (“alg":"none”) or weak secrets (try offline cracking if token uses HMAC). For example, use jwt-cli to change the alg to “none” and re-sign; if the API accepts it, that’s an auth bypass. While not a single tool, this technique is crucial and can be semi-automated with scripts.  
  
authz0 – A tool by ProjectDiscovery (if publicly released) to test Authorization. It can replay captured requests with different tokens or IDs to see if access is improperly granted. Usage: Provide it a collection of requests and it will swap IDs, etc., to look for IDOR (Insecure Direct Object References). This helps find unauthorized access vulnerabilities where one user can access another’s data.  
  
Hoppscotch – An open-source API development client (alternative to Postman). It can script sequences of requests. You can configure it to log in as a low-privileged user and then attempt admin endpoints. Combined with environment variables for tokens, this can systematically test each API call with different roles.  
  
Input Fuzzing & Enumeration (API)  
  
Kiterunner – A tool for discovering hidden API endpoints by bruteforcing common REST paths (especially useful for large microservice APIs). It uses wordlists of API routes (like /api/v1/users, /api/v1/admin). Usage: kiterunner -u https://api.target/ -w api-endpoints.txt -x 401,403 to find endpoints that might exist (including ones not documented). Any “hits” with status 200 could indicate functionality that needs testing (often dev/test endpoints).  
  
Arjun – Automatically finds HTTP GET and POST parameters by sending a huge list of common param names and detecting differences in responses. For APIs, Arjun can discover undocumented parameters (like a hidden ?admin=true or ?debug=). Usage: arjun -u https://api.target/endpoint – if it finds parameter names that cause significantly larger responses or different behavior, you might have discovered a vulnerability trigger (like a debug mode enabling RCE or info leak).  
  
GraphQL Voyager/GraphiQL – If the API uses GraphQL, introspection can leak the entire schema. Open-source GraphQL IDEs or the graphql-introspection tool can query the schema ({"query":"{\_\_schema{types{name fields{name}}}}"}) to enumerate all queries and mutations. From there, you can identify sensitive ones to test (e.g. admin-only mutations). Tools like GraphQLmap can fuzz GraphQL queries for common issues (authorization, injections). Always check if introspection is enabled in production – it’s a minor config issue but leads to big data exposure.  
  
Data Leakage & Misconfigurations (API)  
  
ScoutSuite (Cloud) – If the API is cloud-hosted (AWS/Azure/GCP), ScoutSuite can audit the cloud config for things like open API Gateway endpoints, misconfigured CORS, or publicly exposed databases behind the API  
helpnetsecurity.com  
helpnetsecurity.com  
. It gives a high-level view of cloud resource security. For example, ScoutSuite might flag an S3 bucket that the API uses as public, or an overly permissive IAM role that could be abused  
helpnetsecurity.com  
.  
  
CORS Misconfig Scanner – A simple Python tool or Burp extension can check Cross-Origin Resource Sharing on APIs. It sends requests with various Origin headers to see if the API incorrectly allows all origins (\*) or reflects the origin without proper credentials. Misconfigured CORS can lead to other domains executing API calls on a user’s behalf (data leak or unauthorized actions).  
  
HTTP Headers / TLS – Use testssl.sh (as mentioned) or SSLyze on API endpoints (which might not be on standard port 443). This identifies weak SSL on API servers (a “minor configuration” issue, but important for securing data in transit). Also, tools like securityheaders.com (or local equivalent) can be used on API endpoints to see if security headers (CSP, HSTS, etc.) are set when applicable.  
  
Fuzzing Tools – ffuf and wfuzz aren’t just for directories; they can fuzz JSON and GraphQL inputs too. For instance, fuzz an integer ID in an API path: ffuf -u https://api/objects/£ -w ids.txt to see if you can access others’ data (IDOR). Or fuzz boolean parameters (isAdmin=true/false) in requests to detect privilege escalation.  
  
Smart Contracts (Solana)  
  
Solana smart contracts (called Programs) run on-chain, handling crypto and logic. Vulnerabilities include logic flaws, overflow, or missing checks rather than web bugs. Key open-source tools:  
  
Smart Contract Flaws & Logic Bugs  
  
Solana X-Ray – A newly open-sourced static analyzer from Solana Foundation, built on LLVM, that examines Solana Rust code for vulnerabilities  
solanacompass.com  
solanacompass.com  
. It can detect issues like buffer overflows, arithmetic overflow, missing ownership checks, etc., without running the code  
solanacompass.com  
solanacompass.com  
. Usage: X-Ray can be run via a Docker image or CLI on your Anchor/Rust project; it converts code to an IR and applies security rules. For example, it will flag if you forget to check the signer of a transaction or if arithmetic operations might overflow  
solanacompass.com  
solanacompass.com  
. As an open tool, developers can add custom rules to catch project-specific logic flaws. Docs:   
solanacompass.com  
solanacompass.com  
  
Solana Static Analyzer (Rust) – Solana Analyzer by scab24 – Another Rust static analysis tool for Solana programs  
github.com  
. It scans Anchor/Solana code for common security issues and code quality problems  
github.com  
. Usage: Run it on your program’s source (cargo run -- --path src/ --analyze) to get a report. It includes rules like detecting use of the insecure unsafe Rust code, missing signer or owner verifications, and other high-severity issues (e.g., not checking for overflow if not using SafeMath)  
github.com  
github.com  
. This helps catch vulnerabilities before deployment. Source:   
github.com  
  
Cargo Audit (Rust) – Many Solana programs use Rust crates. cargo-audit checks the project’s Cargo.toml dependencies for known vulns (RustSec Advisory DB) and even can suggest fixes. Usage: In the program directory, run cargo audit: it will list any crate (e.g. a cryptography library) that has a security advisory  
blog.rust-lang.org  
. This ensures no known-vulnerable libraries (which could enable attacks) are used. It’s essentially OWASP Dependency-Check for Rust. Notably, cargo-audit’s --fix can even auto-update some insecure dependencies  
shnatsel.medium.com  
– assisting with patching the contract’s dependencies.  
  
Soteria (Solana Security Toolkit) – A set of open-source scripts and guidelines for Solana program security (if available, e.g., from Solana Labs). It might include checks for ensuring a program is using correct checks (like no unwraps on account data that could panic, etc.). Usage: Follow any included checklist or run provided scripts against the program’s bytecode to highlight risky patterns.  
  
Anchor Security Checks – If the contract is written in Anchor (Solana’s framework), use Anchor’s built-in checks and unit tests. Usage: anchor test with additional tests for edge cases (e.g., multiple signers, overflow values) can catch logic bugs. Also, the community might have open-source linters for Anchor (ensuring no use of &mut ctx.accounts.some\_account without proper constraints).  
  
Secrets & Key Leakage  
  
TruffleHog & Gitleaks – Use these to scan the smart contract repository for any hardcoded private keys, mnemonics, or API secrets  
trufflesecurity.com  
github.com  
. Smart contracts themselves usually shouldn’t contain secrets (they operate with public data), but the deployment scripts or config might. For example, scanning the repo could catch a accidentally committed wallet private key used for testing – critical to remove before an attacker finds it. Docs:   
trufflesecurity.com  
github.com  
  
Secret Detection in On-Chain Data – Tools are emerging to scan Solana account data for known patterns (like private keys accidentally uploaded). While not mainstream, consider custom scripts or searching the ledger for base58 patterns of private keys if relevant.  
  
Dependency & Patch Management (Solana)  
  
Solana Program Update Checker – Ensure your Solana CLI is up-to-date (solana release upgrades often fix security issues). Usage: solana -V and compare to latest release. Keep the Solana SDK and Anchor framework updated in Cargo.toml, as updates may patch vulnerabilities in runtime or macros.  
  
Automated Testing – Not exactly a tool, but writing fuzz tests with something like cargo-fuzz on critical contract functions can expose crashes or panics (which in Solana could mean faults). This is similar to fuzzing native code – any panics in program logic could be exploitable. Open-source crates exist to fuzz test Anchor instructions under a local cluster.  
  
Manual Review and Known Patterns – Leverage community resources like the Awesome Solana Security list (open-source on GitHub) which enumerates common vulnerability patterns (e.g., not using Anchor’s #[account(mut)] correctly leading to wrong account modifications). This isn’t a tool, but a knowledge base that can guide what to test for. For instance, reentrancy isn’t an issue in Solana like EVM, but rent exemption checks, unchecked account privileges, or bypassing seeds are – and static tools above help catch these.  
  
Decentralized Compute  
  
AIxBlock’s decentralized compute implies nodes running workloads (possibly containers or VMs). Vulnerabilities here include container escapes, remote code execution on worker nodes, and misconfigurations. Key tools:  
  
Container and Host Vulnerabilities  
  
Trivy – A comprehensive scanner for container images, file systems, and Kubernetes clusters  
github.com  
github.com  
. It finds OS package CVEs, language library vulns, misconfigurations (Docker/K8s), and even secrets  
github.com  
github.com  
. Usage:  
  
Scan a Docker image: trivy image aixblock/compute-node:latest – outputs any known CVEs in the image’s OS or app packages  
squareops.com  
.  
  
Scan file system: trivy fs ./ on the node’s filesystem or container filesystem to catch leftover creds or keys.  
  
K8s scan: trivy k8s --report summary cluster audits a Kubernetes cluster for config issues (e.g., privileged containers, anonymous access)  
github.com  
.  
  
Trivy also checks IaC (Infrastructure-as-Code) for misconfigs like open security groups. This helps ensure compute clusters and nodes are hardened and patched. Docs:   
github.com  
github.com  
  
Grype – An open-source scanner by Anchore that identifies vulnerabilities in container images and filesystems (similar to Trivy). Usage: grype aixblock/agent:tag will list CVEs in that container image. Use this as part of CI to ensure new images don’t introduce critical vulns. It pulls data from multiple sources to flag high-severity issues that need patching.  
  
Kube-bench – Checks a Kubernetes cluster (if AIxBlock uses k8s for compute orchestration) against CIS Benchmark. It will flag insecure settings (like permissive RBAC roles, non-encrypted secrets, etc.). Usage: kube-bench --cluster and review the report for “Fail” items to fix. This addresses minor config issues that can become severe (like allowing Privileged containers could lead to host RCE).  
  
Kube-hunter – Actively hunts for vulnerable k8s configurations from an attacker perspective (open dashboard, etc.). Usage: Run kube-hunter inside or outside the cluster; it will attempt common exploits. Any findings (e.g., an open etcd server or insecure kubelet port) indicate areas to patch (close ports, enable auth, etc.).  
  
Lynis (Host Audit) – Running Lynis on the actual compute node OS will highlight weak points: outdated kernel (local priv. esc. risk), world-readable sensitive files, SSH config issues, etc.  
tuxcare.com  
. This ensures the base OS where containers run is patched and secure  
linux-audit.com  
. After scanning, remediate per Lynis suggestions (e.g., enable AppArmor, apply kernel updates).  
  
OpenVAS – A free, open-source vulnerability scanner for networks and OS. It can be used to scan the compute node’s IP for open services and known CVEs (similar to Nessus). Usage: Launch OpenVAS, update feeds, then target the node’s IP. It will report any high-risk findings (e.g., vulnerable SSH version, open database service with default creds). Use these results to prioritize patching (CVSS v3.1 criticals first).  
  
Unauthorized Access & Secrets in Compute  
  
ScoutSuite & Prowler – If the decentralized compute leverages cloud instances (AWS/GCP/Azure for nodes), these tools audit the cloud config:  
  
ScoutSuite: Multi-cloud audit tool that checks for things like open SSH security groups, improper IAM roles, public storage buckets, etc.  
helpnetsecurity.com  
. Usage: scoutsuite aws (or the equivalent for the cloud) to generate an HTML report. It might flag, for example, that an S3 bucket with compute node snapshots is public or an IAM role used by nodes has wild card permissions – both would be critical to fix  
helpnetsecurity.com  
helpnetsecurity.com  
.  
  
Prowler: Focuses on AWS (and now multi-cloud) security best practices  
github.com  
github.com  
. It has hundreds of checks for AWS config  
github.com  
. Usage: prowler -c critical to run only critical controls across the account. This will identify misconfigurations like EC2 instances without patches, S3 buckets or ECR images that are public, etc. Prowler’s output helps address high-severity cloud config issues (aligns with standards like CIS AWS foundations)  
github.com  
. Docs:   
github.com  
github.com  
  
Secrets Scanning – Use TruffleHog on any code or config related to the compute network (infrastructure-as-code, deployment scripts). E.g., scanning Terraform scripts that deploy the compute might reveal hardcoded credentials (like cloud API keys)  
qualt.io  
qualt.io  
. Remove or rotate any such secrets. Similarly, if nodes are configured via environment variables, ensure no secrets are exposed in images (Trivy’s secret scanner or Gitleaks can catch these  
github.com  
github.com  
).  
  
Network Scanning (Nmap) – Treat the decentralized compute as an untrusted network: use Nmap to map open ports on compute nodes or associated services. Usage: nmap -p- -sV node-ip to identify any services that shouldn’t be exposed (e.g., an admin panel or Docker API open). Any unexpected service = potential unauthorized access point; secure or firewall it. Nmap NSE scripts can test for known vulns (e.g., nmap --script vuln -p 2375 node-ip could find Docker API without auth, which is critical to fix).  
  
Falco (Runtime Security) – An open runtime threat detection tool that can be deployed on nodes to monitor suspicious behavior (e.g., a container spawning a shell or reading /etc/passwd). While not a finder of existing vulns, Falco helps mitigate and detect exploit attempts in real-time, complementing the above proactive tools. It’s something AIxBlock could use to get alerts if someone manages to execute code on a node (indicating a need to patch that vulnerability ASAP).  
  
Data Engine  
  
“Data Engine” likely refers to data storage or processing components (databases, data pipelines, etc.). Key concerns: data leakage, unauthorized data access, and misconfigurations (like open data stores or inadequate encryption).  
  
Data Storage Vulnerabilities & Leakage  
  
Mongoaudit – An open-source MongoDB auditing tool that checks for poor settings and known vulns  
github.com  
bleepingcomputer.com  
. Usage: mongoaudit --host <db-host> will report issues: e.g., no auth on the DB, default admin credentials, outdated MongoDB version, etc.  
github.com  
. It even gives fix advice  
bleepingcomputer.com  
. This helps secure NoSQL databases by identifying misconfigurations (which have led to many breaches). Docs:   
github.com  
medium.com  
  
SQLmap (again) – If the data engine is exposed via an API or web, SQLmap can exploit SQL injections to dump data  
kali.org  
kali.org  
. Use it to identify any input that can directly query the data store. The goal is to find and fix injections (sanitize inputs, use ORM parameterization, etc.). For patching: after using sqlmap to confirm an injection, developers can apply proper query parameterization or stored procedures to eliminate the flaw.  
  
S3Scanner – Many data leaks come from cloud storage buckets. S3Scanner finds open S3 buckets across AWS and other providers  
github.com  
. Usage: s3scanner -bucket-file names.txt (with a list of bucket names or patterns) checks each for public access  
github.com  
github.com  
. It reports if buckets exist and if you can list or read their content. This directly identifies publicly accessible data that should be private. For any “open” bucket found, the fix is to update its ACL/policy (e.g., restrict to auth users or specific IPs). Source:   
github.com  
  
Cloud Storage Suite – Similar tools exist for other clouds (e.g., GCP Bucket Brute for Google Cloud Storage, or built-in commands like gsutil ls -al gs://bucket). Use these to ensure no storage is unintentionally public. Prowler (mentioned prior) also includes checks for S3 buckets being public or not encrypted at rest  
reddit.com  
.  
  
Elasticsearch/Redis Scanners – If the data engine includes search or caching services, there are simple Python scripts or Nmap NSE modules to check if these are open:  
  
Example: nmap -p 6379 --script redis-info <host> will tell if a Redis instance is unprotected (no auth, and info leakage). If yes, that’s a critical issue (attackers could dump or manipulate cache). Fix by enabling Redis AUTH and binding to localhost.  
  
For Elasticsearch, tools like ScoutSuite will flag if the cluster permits unauthorized access or if X-Pack security is disabled.  
  
Unauthorized Data Access & API  
  
Autoswagger / Postman – Use the Autoswagger approach on data-related APIs too (as described, to catch endpoints exposing data without auth)  
helpnetsecurity.com  
helpnetsecurity.com  
. Also, with Postman, test role-based access: e.g., a regular user calling an admin-only data export API. If the API does not properly check roles, that’s an authorization bypass to report and fix (by enforcing server-side role checks).  
  
Burp Suite Community + Extensions – Free Burp with the AuthMatrix extension can systematically check access control. Set up user contexts (admin, user, etc.) and let AuthMatrix attempt each endpoint with each user’s token. It will quickly highlight any response anomalies (like a normal user successfully accessing an admin endpoint). Though this is partly manual, it’s using an open extension to reveal unauthorized data access paths.  
  
Data Masking Checks – Custom scripts can search responses for sensitive data patterns (e.g., PII like SSNs, credit card numbers). For instance, after running a spider or using ZAP to passively capture API responses, you could grep for patterns (using grep or Python) to detect leakage of things like private keys or personal data that shouldn’t be exposed. If found, that indicates insufficient data masking or access control.  
  
Configuration & Remediation  
  
Prowler & ScoutSuite – Emphasizing these again for data layer: they will check if databases have backups publicly exposed, or if encryption is enabled. For example, Prowler will warn if an RDS database instance isn’t using encryption at rest, or if an Elasticsearch domain is open to the internet. All these align with CVSS high severity if exploitation is possible (e.g., a public DB with no auth is Critical). Use their output to guide remediation: enable encryption, restrict network access, enable auditing logs, etc.  
github.com  
helpnetsecurity.com  
.  
  
Backup and Artifact Scans – Tools like gitrob (open-source) can find sensitive files in GitHub repos (e.g., database dumps or API keys). If the data engine code or config is in a repo, run gitrob or similar to ensure no dev accidentally committed a data snapshot. Removing such artifacts prevents attackers from leveraging leaked data.  
  
Dependency Updates – If the data engine uses frameworks (Hadoop, Spark, etc.), use their equivalent of dependency-check. For example, run pip-audit if Python libraries are used in data processing – known vulnerable versions (like an outdated Pandas with RCE CVE) should be upgraded. Similarly, use npm audit or gradle audit if applicable. Patch management tools (even simple bash scripts) can track when a critical CVE (CVSS 9.0+) is announced for your data software and prompt immediate updates.  
  
Webhooks  
  
Webhooks are HTTP callbacks – security issues include weak validation (allowing spoofed calls), leaking secrets or accepting malicious payloads. Tools and methods:  
  
Webhook Tester (tarampampam/webhook-tester) – An open-source tool to simulate endpoints and inspect webhook calls  
github.com  
. Use it to ensure your webhooks include a secret/token. Usage: Deploy webhook-tester locally or use a service like webhook.site, then configure the real service to send webhooks to your test URL. Observe the headers and payload:  
  
If there’s no signature header or authentication, that’s a vulnerability (attackers could send fake webhooks).  
  
If the payload contains sensitive data unencrypted, that might be an issue (depending on context).  
This isn’t an automated scanner, but it helps verify proper webhook implementation (a misconfiguration if secrets aren’t used).  
  
Open Redirect & SSRF Checks – If the webhook URL is user-supplied in AIxBlock (for example, users give a callback URL), test for SSRF: provide an internal URL (like http://localhost/admin) as the webhook target and see if AIxBlock’s system tries to fetch it. Tools like SSRFmap can assist by generating a variety of URL payloads (file://, gopher://). If internal content is returned to you or an external canary domain is hit, there’s an SSRF via webhook mechanism – needs to be patched by validating outgoing webhook URLs.  
  
cURL and OpenSSL s\_client – These basic tools can be used to simulate webhook requests to your endpoint to test client-side TLS config:  
  
openssl s\_client -connect your-webhook-endpoint:443 to see if the webhook client uses a specific TLS version or cipher. (This is more relevant to ensure the caller (AIxBlock) uses strong TLS when calling out, a configuration detail often overlooked.)  
  
Use cURL to replay webhook payloads at the endpoint without the expected signature to confirm it’s rejected. e.g., curl -X POST -d '{"test":1}' https://your.service/webhook without auth and see if it processes. It should not – if it does, implement token checking.  
  
Security Header Check – Ensure the webhook endpoints (the receivers) enforce good practice (like requiring X-Hub-Signature or similar). Tools won’t directly tell you this, but reviewing documentation or using the above test is key. If the program’s webhooks come from known sources (e.g., GitHub webhooks), you can use their IP ranges to whitelist; scanning your firewall with Nmap can verify only those ranges are allowed.  
  
Minor Issues: For example, if webhooks re-use IDs or predictable tokens, an attacker might brute-force them. Use a fuzzer (ffuf) to try various IDs at a webhook receive endpoint if the URL contains an identifier. Lack of entropy = needs a fix (use more randomness).  
  
In summary, for webhooks a lot is manual: using available tools to simulate and verify that authentication (usually HMAC secrets) and origin checks are in place. Once identified, issues like “webhook endpoint accepts unsigned calls” should be mitigated by requiring a secret token and verifying it on receipt.  
  
MCP Integration Layer  
  
The “MCP Layer” connects AIxBlock with third-party systems. It likely involves APIs, SDKs, or bridging networks. Major focus: secure configurations, no open ports, and up-to-date dependencies:  
  
Nmap & OpenVAS – Use Nmap to scan the MCP integration interface for any open ports or services that shouldn’t be exposed (similar to the earlier network scans). For instance, if MCP listens on a port for incoming connections, ensure only intended hosts can reach it (use Nmap from outside and inside the network). OpenVAS can perform a deeper scan on those services for known vulns. For example, if MCP Layer runs an MQTT broker (just guessing an integration tech), OpenVAS might find default creds or a CVE in that broker – which should then be patched or access-controlled.  
  
TestSSL.sh – If the MCP layer uses secure communication (e.g., mutual TLS with partners), run testssl.sh against it to ensure modern protocol usage  
kali.org  
. This will flag any outdated TLS or weak ciphers so you can disable them  
kali.org  
. A strong integration requires strong transport security (aligns with Integrity/Confidentiality best practices).  
  
Packet Analysis Tools – If possible (with permission), capture MCP traffic using Wireshark or tcpdump. This can reveal if sensitive data or tokens are sent in the clear. While not an “automated vuln scanner,” it’s invaluable for ensuring no information leakage in integration flows. If you see something like an API key being consistently reused or sent without encryption, that’s a critical issue to fix (use short-lived tokens or encrypt the channel).  
  
Dependency and Build Scans – The MCP layer might be a piece of middleware (perhaps a Node.js or Java service). Leverage SAST/DAST for it:  
  
Run OWASP Dependency-Check on the MCP codebase to catch any vulnerable libraries  
github.com  
.  
  
If it’s Java, consider FindSecBugs (an open static analysis plugin) to find common mistakes (e.g., trustAll SSL context, which would be bad in integration security).  
  
If it’s Python, Bandit can find issues (like using subprocess unsafely if that’s part of integration, or hardcoded credentials).  
  
After identifying issues, update those libraries (patch) or change code (e.g., do not ignore SSL certificate validation).  
  
Audit Logs & Access – Check that the MCP layer has proper logging and access control. While not a tool, you can use scripts to attempt accessing MCP internal endpoints without auth. For example, if MCP has an admin panel at /mcp/admin, use a tool like Gobuster to find it, then see if it’s protected. Lack of auth on integration admin interfaces is a critical flaw.  
  
Configuration Linters – If MCP uses config files (YAML/JSON), tools like checkov (open-source IaC scanner) might catch risky settings. For example, checkov can parse Kubernetes or Terraform files for integration deployments and warn if secrets are in plain text or if secure parameters (like allowed CIDRs) are too open.  
  
Remediation Tools: Once issues are identified in MCP:  
  
Use package managers to update vulnerable components (npm/yarn, pip, etc., as appropriate).  
  
Implement firewall rules (you can test these with Nmap: after applying a rule, Nmap from a blocked IP should show port closed).  
  
Incorporate a CI pipeline with these tools (many like Trivy, Dependency-Check, Bandit can run in CI) so that future changes to MCP are automatically scanned for high-severity problems.  
  
Finally, ensure all findings align with CVSS v3.1 high/critical criteria. For instance:  
  
RCE or SQLi = Critical (use scanners like ZAP, Wapiti, sqlmap to find, then patch by sanitization and updates).  
  
Auth bypass or IDOR = High/Critical (use Autoswagger, AuthMatrix to find, then fix by adding proper checks).  
  
Public S3 bucket or open DB = Critical (use S3Scanner, Mongoaudit, then restrict access and enable auth/encryption).  
  
Outdated component with known RCE = High (use Dependency-Check or Trivy, then upgrade dependency).  
  
Misconfigurations like missing HTTPS, weak TLS, or missing CSRF tokens are lower severity individually, but still should be fixed as Defense in Depth.  
  
By systematically using the above open-source tools in each domain, one can identify vulnerabilities and then patch/remediate them (via updates, config changes, or code fixes) to secure the AIxBlock platform. Each tool’s documentation (linked above) provides more detail on usage and mitigation recommendations so you can not only find issues but also resolve them in line with security best practices.  
  
does all the work above (get entire chat context) prescribe to this -   
  
Great, lets please review the below to see whether any other vulnerabilities can be found and whether additional depth can be gained.  
  
@https://www-cdn.anthropic.com/6be99a52cb68eb70eb9572b4cafad13df32ed995.pdf   
  
@https://cybersecuritynews.com/hackers-weaponize-aws-x-ray-service/   
  
@https://cybersecuritynews.com/ransomware-gangs-leverage-remote-access-tools/   
  
@https://github.com/sensepost/wpswag   
  
@https://github.com/sensepost/   
  
Use other resources mentioned to gain clarity or additional information.