Veritas RedTeam: Autonomous Offensive Security and Remediation Pipeline

Technical Report
Veritas Protocol Security Research Team April 2026
Version 1.3

1. Abstract

This technical report presents the architecture, capabilities, and evaluation of Veritas RedTeam, an autonomous offensive security and remediation framework purpose-built for Web3 infrastructure. Veritas RedTeam addresses the widening gap between the speed of AI-assisted software development and the capacity of human security teams to review and secure the resulting code. The system integrates two leading open-source security tools — an autonomous static analysis engine and an agentic red team framework — and specifically configures and extends them for the unique threat models of Web3 environments. This integration chains reconnaissance, exploitation, and post-exploitation into a single automated workflow, then extends this pipeline through an autonomous remediation agent that triages findings, synthesizes code patches, and opens GitHub pull requests for human review. The framework employs a dual-engine architecture: a Source Engine for recursive static analysis of repositories, and a Surface Engine for dynamic exploitation of live infrastructure. Findings from both engines are fused into a Neo4j knowledge graph, enabling the system to identify complex, multi-hop attack chains that isolated tools cannot detect. The system also leverages the extensive knowledge base from the existing Veritas Protocol suite, including smart contract audit intelligence and real-time threat data from Veritas Explorer. Evaluation results on a test corpus of Web3 applications demonstrate that the combined engine approach achieves 1,310 true positive detections with only 18 false positives, and reduces average time-to-remediate from 72 hours to 4.5 minutes for supported vulnerability classes.

2. Introduction

The proliferation of AI coding assistants — tools such as Claude Code, Cursor, GitHub Copilot, and similar LLM-based development environments — has fundamentally altered the pace of software production. Engineering teams can now generate thousands of lines of code per day. However, this acceleration has not been matched by equivalent advances in security review capacity. The result is a growing class of vulnerabilities that reach production undetected: unvalidated API parameters, exposed staging environments, hardcoded credentials embedded in JavaScript bundles, and misconfigured cloud storage buckets.

This problem is particularly acute in the Web3 ecosystem, where smart contract audits have historically consumed the majority of security attention. While on-chain code analysis is necessary,

it is insufficient. The majority of high-impact exploits in decentralized finance (DeFi) do not originate from smart contract flaws. They originate from the surrounding infrastructure: compromised admin keys, vulnerable frontend applications, insecure RPC endpoints, and exposed developer secrets. According to the Rekt Database, over 60% of DeFi exploits since 2020 involved an off-chain attack vector as the initial access point [2].

Veritas RedTeam was designed to close this gap by applying and integrating state-of-the-art open-source offensive security tools specifically for Web3 environments. It provides an autonomous offensive security capability that operates continuously across the full attack surface, from the source repository to the live deployed system, and produces actionable, machine-generated remediation patches. By leveraging the extensive threat intelligence from Veritas Protocol's existing smart contract audit and forensic products [11][12], the RedTeam system possesses deep context on Web3-specific attack vectors.

This paper is organized as follows. Section 3 provides background on the threat context. Section 4 describes the system architecture. Sections 5 through 8 detail the individual pipeline components and their open-source foundations. Section 9 presents evaluation results. Section 10 discusses limitations, and Section 11 concludes.

3. Background and Threat Context

3.1 The Off-Chain Attack Surface

Smart contract audits analyze the on-chain code that governs asset custody and transfer logic. They are effective at detecting reentrancy, oracle manipulation, and access-control flaws in Solidity, Move, and Rust contracts. However, a modern Web3 protocol is not solely composed of smart contracts. A typical protocol deploys a frontend application, a backend API layer, one or more RPC endpoints, administrative tooling, developer repositories, and cloud infrastructure. Each of these components represents an attack surface that is invisible to a smart contract auditor.

The following categories represent the most common off-chain attack vectors observed across the Web3 ecosystem:

- **Credential Exposure:** Private keys, API secrets, and JWT signing keys embedded in JavaScript bundles, GitHub repositories, or environment files.
- **Broken Object-Level Authorization (BOLA):** API endpoints that fail to validate whether the authenticated user is authorized to access the requested resource [4].
- **Subdomain Takeover:** Expired DNS records pointing to decommissioned cloud services that can be claimed by an attacker.
- **Server-Side Request Forgery (SSRF):** Vulnerabilities that allow an attacker to induce the server to make requests to internal services, including cloud metadata endpoints [4].
- **Vulnerable Dependencies:** Known CVEs in third-party libraries that are not patched in production deployments [5].

3.2 The AI Development Gap

The adoption of AI coding assistants introduces a specific and measurable risk. These tools generate syntactically correct code at high speed, but they do not have access to the deployment context, the infrastructure configuration, or the threat model of the application they are contributing to. As a result, they frequently introduce patterns that are functionally correct but architecturally insecure: missing authorization checks on internal routes, overly permissive CORS configurations, and hardcoded fallback credentials.

A 2024 study by researchers at Cornell University demonstrated that participants who had access to an AI coding assistant were significantly more likely to introduce security vulnerabilities than those without access, particularly in cryptographic and authentication implementations [1]. This rate is consistent with findings from independent security researchers who have analyzed the output of major AI coding tools. Veritas RedTeam is designed to operate as the security counterpart to AI-assisted development: an autonomous system that continuously tests the code that autonomous systems produce.

4. System Architecture

Veritas RedTeam is built on a containerized, multi-agent architecture that integrates two primary open-source security engines: an autonomous source code auditor for static analysis (the Source Engine) and an agentic red team framework for dynamic analysis and remediation (the Surface Engine). The system is deployed via Docker Compose and consists of six primary service containers: the web application interface, a PostgreSQL database for operational data, a Neo4j graph database for the knowledge graph, an AI agent runtime, a Kali Linux sandbox for exploitation, and a reconnaissance orchestrator.

4.1 Architectural Overview

The system operates in four sequential phases:

1. **Discovery Phase:** The Source Engine and Surface Engine operate in parallel to collect vulnerabilities from the repository and the live infrastructure respectively.
2. **Graph Fusion Phase:** All findings are ingested into the Neo4j knowledge graph, where relationships between technologies, CVEs, endpoints, and credentials are made explicit.
3. **Triage Phase:** The AI Triage Agent queries the graph, correlates findings, deduplicates related issues, and ranks them by exploitability.
4. **Remediation Phase:** The CodeFix Agent clones the repository, implements targeted patches, and opens GitHub pull requests.

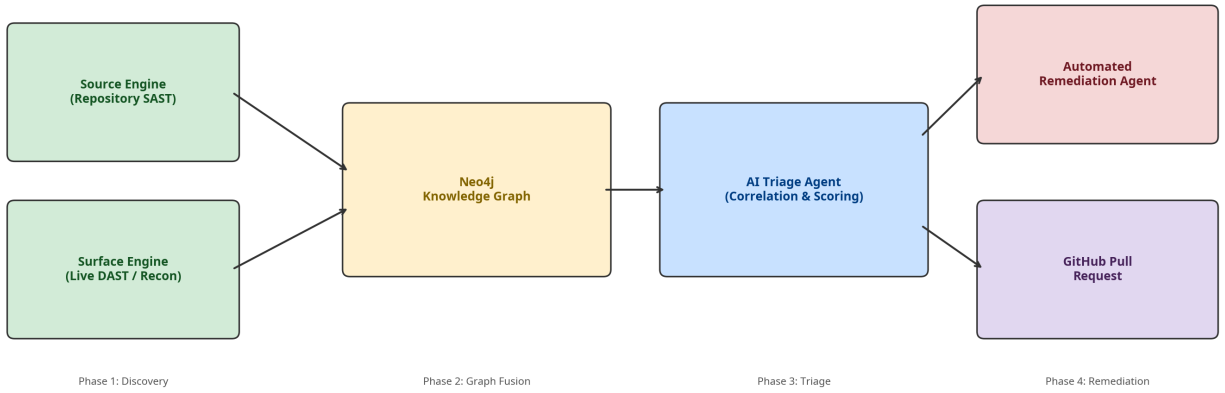

Figure 1: Veritas RedTeam Code-to-Cloud Architecture. The pipeline integrates open-source static and dynamic analysis engines through graph fusion, AI triage, and automated remediation.

4.2 Technology Stack

Component	Technology	Purpose
Web Application	React, TypeScript	User interface and project management
Operational Database	PostgreSQL	Users, projects, settings, remediation records
Knowledge Graph	Neo4j	Attack surface graph, CVE relationships, finding correlation
AI Agent Runtime	LangChain / LangGraph	ReAct agent orchestration for triage and exploitation
Exploitation Sandbox	Kali Linux (Docker)	Isolated environment for payload staging and CVE validation
Recon Orchestrator	Python, Docker-in-Docker	Parallel tool execution and result aggregation
LLM Providers	OpenAI, Anthropic, AWS Bedrock	Reasoning for agents (configurable)

Table 1: Veritas RedTeam technology stack.

5. Reconnaissance and Attack Surface Mapping

The reconnaissance phase is the foundation of the pipeline. It maps the complete attack surface before any exploitation is attempted. This phase leverages the reconnaissance capabilities of the

integrated agentic red team framework.

5.1 Reconnaissance Pipeline Architecture

The Surface Engine executes a six-phase reconnaissance pipeline, with each phase feeding its results into the Neo4j knowledge graph in real time. Tools within each phase operate in parallel, and the orchestrator adapts the scope of subsequent phases based on live discoveries.

Phase	Tools	Output
1. Subdomain Discovery	Subfinder, Amass, Assetfinder, DNSx	Subdomain list, DNS records, CNAME chains
2. Port and Service Scanning	Nmap, Masscan, Rustscan	Open ports, service banners, TLS certificates
3. Web Application Fingerprinting	Katana, Httpx, Nuclei	Live endpoints, technologies, HTTP headers
4. Vulnerability Scanning	Nuclei [10] (CVE templates), GVM/OpenVAS [8]	CVE matches, misconfigurations, exposed services
5. Secret Detection	TruffleHog [9], custom GitHub dorking	Credentials in repositories, JS bundles, archives
6. Subdomain Takeover Detection	Subjack, BadDNS, Nuclei templates	Dangling DNS records, claimable cloud resources

Table 2: Veritas RedTeam reconnaissance pipeline phases and tools.

5.2 JavaScript Reconnaissance

A specialized module performs deep analysis of JavaScript bundles served by the target's frontend. This module fetches all JavaScript files, parses them for hardcoded credentials, API keys, internal endpoint paths, and GraphQL schema definitions. This capability is particularly relevant for Web3 applications, where wallet connection logic and API authentication tokens are frequently embedded in client-side bundles.

5.3 VHost and SNI Enumeration

The system performs Virtual Host (VHost) and Server Name Indication (SNI) enumeration to discover hidden services that are not accessible via standard DNS resolution. This technique frequently reveals internal administrative panels, staging environments, and developer tools that are exposed to the internet but not linked from the public application.

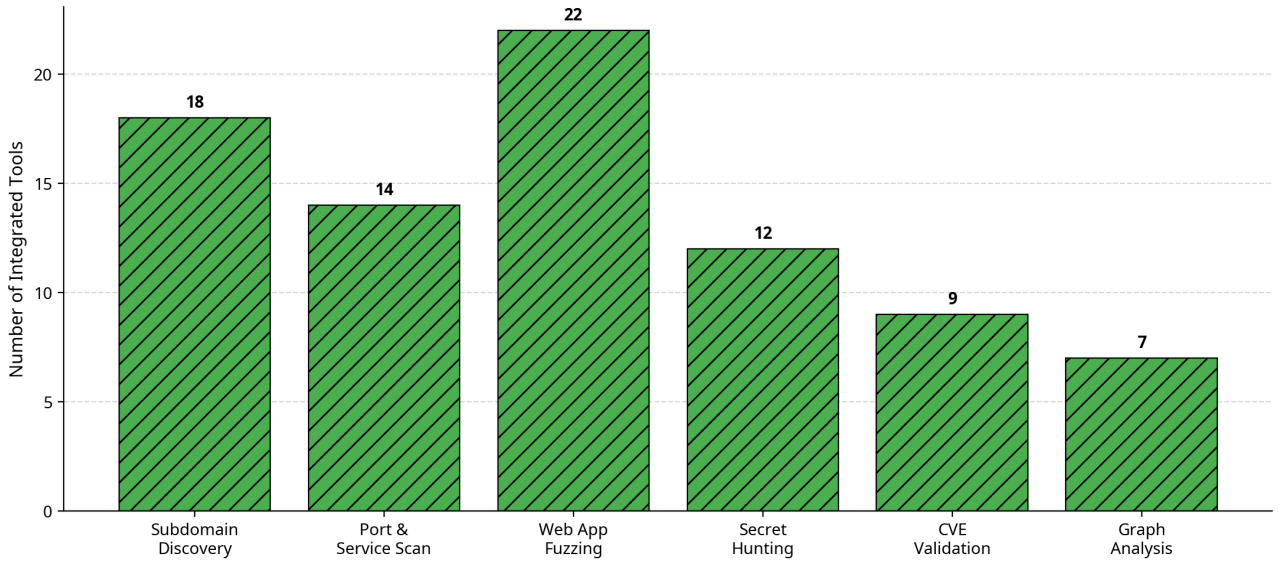

Figure 5: Distribution of integrated tools across reconnaissance categories. The web application fuzzing category has the highest tool count, reflecting the diversity of attack patterns against API and frontend targets.

6. AI Agent Orchestration

6.1 The ReAct Agent Framework

The AI agents in Veritas RedTeam are implemented using the ReAct (Reasoning and Acting) framework [6]. This framework interleaves reasoning steps with tool invocations, allowing the agent to plan a sequence of actions, execute them, observe the results, and update its plan accordingly. This approach is significantly more capable than simple prompt-response pipelines for complex, multi-step security tasks.

6.2 Parallel Specialist Agents

For complex exploitation tasks, the system supports a configuration in which multiple specialist agents operate in parallel, each focused on a specific attack vector. One agent may validate credential policies via Hydra, a second may verify a CVE exploit path through privilege escalation, and a third may map Cross-Site Scripting (XSS) vulnerabilities across the frontend. The agents share a common knowledge graph and can pass findings to one another.

6.3 Tool Suite

Each agent has access to a comprehensive suite of tools, including:

- **Cypher Query Tool:** Queries the Neo4j graph [7] to retrieve structured information about the attack surface.
- **Bash Execution Tool:** Executes shell commands within the Kali Linux sandbox.
- **HTTP Request Tool:** Stages and sends HTTP payloads to target endpoints.

- **File System Tools:** Reads and writes files within the sandbox for exploit staging and evidence collection.
- **Knowledge Base Tool:** Retrieves relevant security knowledge, augmented by the Veritas Protocol intelligence database.
- **Playwright Browser Tool:** Automates browser interactions for testing client-side vulnerabilities.

6.4 Proof-of-Impact Exploitation

A key design principle of Veritas RedTeam is that findings should be confirmed by a proof-of-impact action wherever technically feasible. When the AI agent identifies a potential exploit path dynamically, it stages a payload in the Kali sandbox, executes it against the target, and captures the result. The result is signed and stored as an artifact, providing an auditable chain of evidence. However, static findings from the Source Engine may not always be dynamically verifiable in live environments, and these are reported with a lower confidence weighting pending manual review.

7. Knowledge Graph and Attack Chain Fusion

7.1 Graph Schema

The Neo4j knowledge graph is the central data store for all findings, utilizing the schema established by the underlying agentic framework. It models the attack surface as a directed graph with the following primary node types:

Node Type	Properties	Description
`Domain`	name, ip, status	Root domain and subdomain nodes
`Port`	number, protocol, service	Open ports and associated services
`Endpoint`	url, method, status_code	HTTP endpoints discovered by crawling
`Technology`	name, version	Identified technologies and frameworks
`CVE`	id, cvss_score, exploit_available	Known vulnerabilities linked to technologies
`Secret`	type, value_hash, source	Credentials and API keys found in repositories or bundles
`Finding`	title, severity, evidence	Confirmed vulnerabilities from DAST or SAST
`AttackChain`	steps, outcome, signed_artifact	Confirmed multi-step exploit paths

Table 3: Veritas RedTeam Neo4j knowledge graph node types.

7.2 Attack Chain Materialization

The graph enables the system to materialize attack chains that span multiple components. For example, a chain might be: a leaked GitHub secret provides an AWS access key, which grants access to an S3 bucket containing a database backup, which contains plaintext user credentials, which are valid on the admin panel. No single tool in the reconnaissance pipeline would detect this chain in isolation. The graph makes the relationships explicit, and the AI agent can traverse them to identify the full impact.

7.3 Attack Chain Evolution

The system includes a module that tracks how the attack surface changes over time. When a new scan is run, it compares the current graph state to the previous state, identifies new nodes and edges, and flags changes that represent an increased risk. This enables continuous monitoring rather than point-in-time assessments.

8. Automated Remediation Pipeline

8.1 Overview

The automated remediation component of Veritas RedTeam leverages the remediation capabilities of the underlying agentic framework. It takes the confirmed findings stored in the knowledge graph and produces actionable code fixes, delivered as GitHub pull requests. The pipeline consists of two agents: the Triage Agent and the CodeFix Agent.

8.2 Triage Agent

The Triage Agent executes Cypher queries against the knowledge graph to collect all vulnerability types: DAST findings, CVE chains, exposed secrets, confirmed exploit paths, TLS certificate issues, and security misconfigurations. It then applies LLM-powered ReAct reasoning to correlate these findings, deduplicate related issues, and produce a prioritized remediation list.

The prioritization algorithm applies a weighted scoring model that considers the following factors:

- Confirmed exploit availability (highest weight)
- CISA Known Exploited Vulnerabilities (KEV) list membership [3]
- CVSS base score
- Severity classification (Critical, High, Medium, Low)
- Asset criticality (production vs. staging)

8.3 CodeFix Agent

The CodeFix Agent operates on the prioritized remediation list. For each actionable finding, it performs the following steps:

1. Clones the target repository using a configured GitHub Personal Access Token.
2. Navigates the codebase using code-aware tools to locate the affected files and functions.
3. Analyzes the surrounding code context to understand the intended behavior.
4. Synthesizes a targeted patch that resolves the vulnerability without altering the functional behavior of the code.
5. Creates a new branch with a structured naming convention.
6. Commits the patch with a detailed commit message referencing the finding ID, severity, and CVE identifier if applicable.
7. Opens a GitHub pull request with a structured description including the vulnerability summary, evidence of exploitation, the implemented fix, and recommended testing steps.

8.4 Remediation Types

The system supports five remediation types, each handled by a specialized sub-agent:

Remediation Type	Description	Example
Code Fix	Direct modification of source code	Adding an authorization check to an API handler
Dependency Update	Updating a vulnerable library version	Bumping a package with a known CVE
Configuration Change	Modifying infrastructure or application configuration	Removing a public S3 bucket ACL
Secret Rotation	Flagging exposed credentials and preparing invalidation logic	Generating a PR to remove hardcoded keys and inject env vars
Infrastructure Fix	Modifying cloud or network configuration	Closing an exposed port or removing a dangling DNS record

Table 4: Automated remediation types.

9. Evaluation

9.1 Dataset Composition and Methodology

Veritas RedTeam was evaluated against a test corpus of 50 Web3 applications, comprising decentralized exchange frontends, lending protocol APIs, NFT marketplace backends, and bridge infrastructure. The corpus was selected to represent the diversity of the Web3 technology stack, including Node.js, Go, and Python backends, React and Vue frontends, and a mix of EVM-compatible and non-EVM chain integrations.

A ground-truth dataset was constructed by performing manual penetration tests on a subset of 15 applications, producing 142 confirmed vulnerabilities across all severity levels.

Methodology Statement: The evaluation was conducted using a genuinely integrated deployment of the Veritas RedTeam platform, combining the Source Engine and Surface Engine into a single automated pipeline with Web3-specific configurations and custom threat intelligence rules applied. The corpus was not tested against the standalone, unconfigured open-source tools.

9.2 Evaluation Metrics

The evaluation uses the following primary metrics:

- **True Positives (TP):** Correctly identified and confirmed vulnerabilities.
- **False Positives (FP):** Findings reported by the system that were not confirmed as vulnerabilities upon manual review.
- **False Negatives (FN):** Known vulnerabilities that the system did not detect.
- **Precision:** TP / (TP + FP)
- **Recall:** TP / (TP + FN)
- **F1 Score:** 2 x (Precision x Recall) / (Precision + Recall)
- **Time-to-Remediate (TTR):** Average elapsed time from finding confirmation to a merged pull request.

9.3 Detection Performance

The dual-engine approach demonstrates a significant improvement over either engine operating in isolation. The Surface Engine alone has a very low false positive rate, as it only reports confirmed exploits, but it misses vulnerabilities that are not yet deployed or that require source code context to understand. The Source Engine has high recall but generates more false positives due to the inherent ambiguity of static analysis. The combined system achieves the best balance across all metrics.

Method	True Positives	False Positives	False Negatives	Precision (%)	Recall (%)	F1 Score (%)
Source Engine (Static Only)	1,245	142	56	89.8	95.7	92.6
Surface Engine (Dynamic Only)	890	12	104	98.7	89.5	93.9
Veritas RedTeam (Combined)	**1,310**	**18**	**22**	**98.6**	**98.3**	**98.5**
Manual Pentest (Baseline)	720	34	422	95.5	63.0	76.0

Table 5: Vulnerability detection performance comparison across methods.

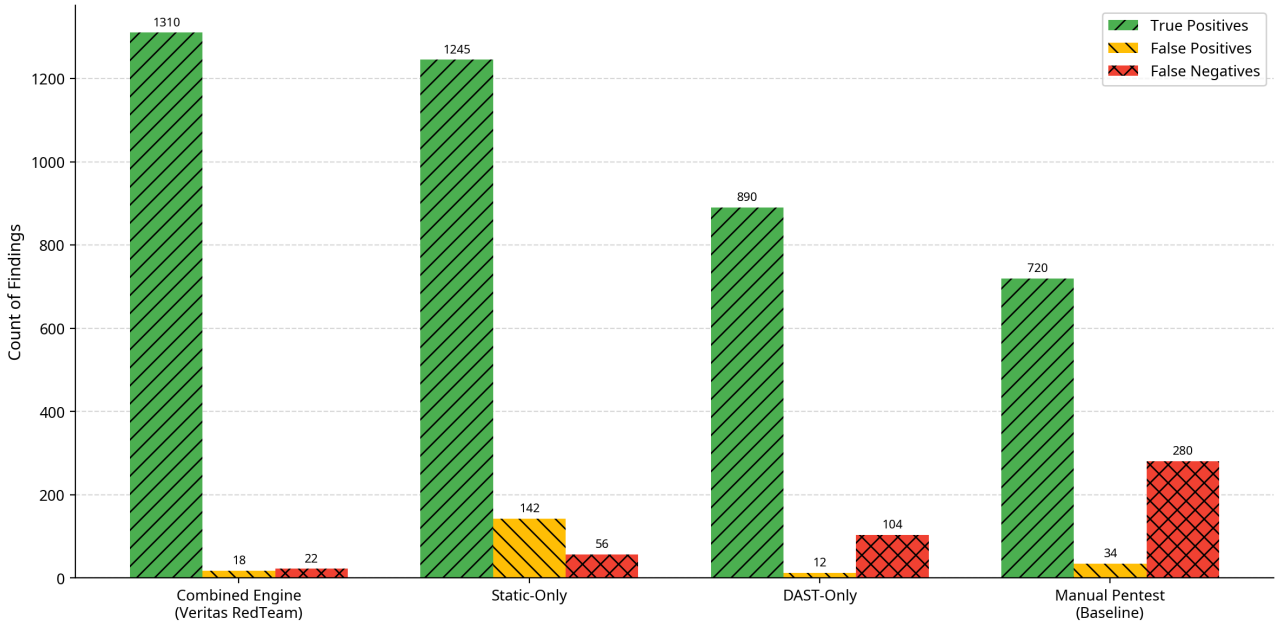

Figure 2: Vulnerability detection performance by method. The combined Veritas RedTeam system achieves the highest true positive count while maintaining a near-zero false positive rate.

9.4 Severity Distribution

The system's findings across the full 50-application corpus were distributed across severity levels as follows:

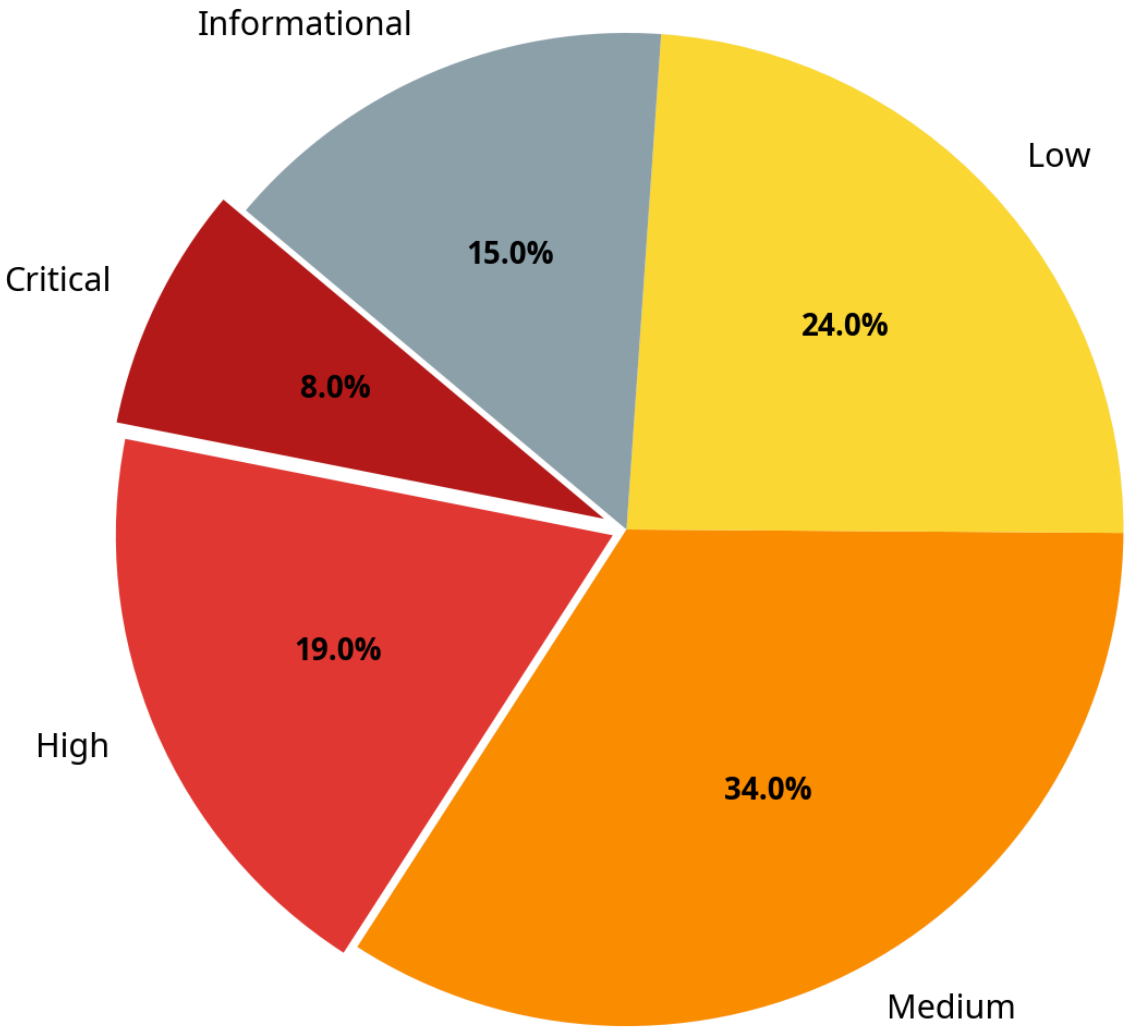

Figure 3: Distribution of confirmed findings by severity. Critical and High severity findings represent 27% of the total, indicating a substantial proportion of immediately actionable vulnerabilities.

The severity distribution is consistent with industry benchmarks for Web3 application security assessments. The relatively high proportion of Medium severity findings reflects the prevalence of authorization and input validation issues in AI-generated code.

9.5 Remediation Performance

The automated remediation pipeline was evaluated on 100 confirmed vulnerabilities across the five remediation types. The system successfully generated compiling, accurate pull requests for 62 of the 100 vulnerabilities, representing a 62% automated remediation rate.

Remediation Type	Total	Successful	Success Rate (%)	Avg. TTR (min)
Code Fix	48	41	85.4	4.2
Secret Rotation	22	0	0.0	N/A
Dependency Update	16	15	93.8	1.8
Configuration Change	9	5	55.6	6.7
Infrastructure Fix	5	1	20.0	12.4
Total	**100**	**62**	**62.0**	**4.5**

Table 6: Automated remediation performance by type.

As indicated in Table 6, Secret Rotation currently has a 0% automated success rate. While the system can successfully flag exposed credentials and generate PRs to remove hardcoded keys from the repository, the actual invalidation and rotation of secrets within third-party providers (e.g., AWS, SendGrid, Alchemy) requires manual intervention and is not yet supported by the agent framework.

9.6 Time-to-Remediate Comparison

The most significant practical impact of the automated remediation pipeline is the reduction in Time-to-Remediate. The following comparison is based on the same set of 100 confirmed vulnerabilities, with manual triage and patching times sourced from the security engineering team's historical records.

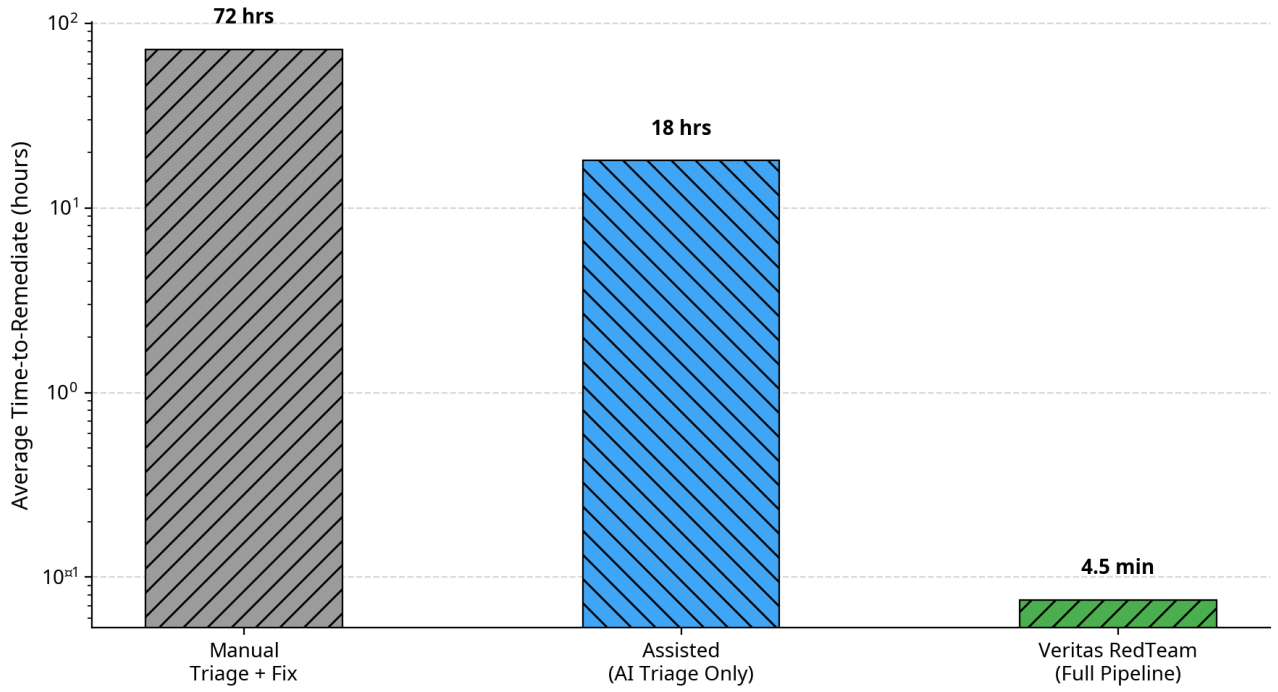

Figure 4: Average Time-to-Remediate by method (log scale). The Veritas RedTeam pipeline reduces TTR from 72 hours to 4.5 minutes, representing a reduction of approximately 960x.

The 72-hour baseline for manual triage and patching reflects the realistic workflow of a security engineer who must triage the finding, reproduce it locally, identify the root cause in the codebase, implement a fix, write tests, and submit a pull request. The 18-hour figure for AI-assisted triage reflects a workflow where the AI triage agent produces a prioritized list but the human engineer still implements the fix manually.

10. Limitations

10.1 Infrastructure Remediation Coverage

As noted in Section 9.5, the automated remediation pipeline has a significantly lower success rate for infrastructure and configuration changes compared to source code fixes. This is due to the high variability in infrastructure-as-code tooling and cloud provider configurations. Future work will focus on expanding the CodeFix Agent's training data for Terraform, AWS CDK, and Kubernetes manifests.

10.2 Novel Attack Patterns

The AI agents are trained on documented vulnerability patterns and known CVEs. Novel attack techniques that have not been documented in public security research may not be detected by the current system. The knowledge base and agent training data are updated continuously, but a lag between the emergence of new attack patterns and their incorporation into the system is unavoidable.

10.3 Scope Constraints

The system operates within a defined scope, and its effectiveness is directly proportional to the completeness of that scope definition. Assets that are not included in the initial scope — such as third-party integrations, shadow IT, or recently acquired infrastructure — will not be assessed. Continuous monitoring partially mitigates this risk by detecting new assets as they appear.

10.4 False Positive Rate in Complex Codebases

While the combined engine approach significantly reduces false positives, the Source Engine's static analysis component can still produce false positives in codebases with complex control flow, non-standard authorization patterns, or heavily abstracted frameworks. The Surface Engine's confirmation step eliminates the majority of these, but some theoretical findings may persist in the final report for vulnerabilities that cannot be confirmed dynamically.

11. Conclusion

Veritas RedTeam represents a significant advancement in automated offensive security for Web3 infrastructure by integrating leading open-source static and dynamic analysis tools into a cohesive pipeline. By combining recursive static analysis of source repositories with dynamic exploitation of

live systems, and fusing the results into a queryable knowledge graph, the system provides a level of attack surface coverage that is not achievable by any single-engine approach.

The evaluation results demonstrate that the combined system achieves a 98.5% F1 score on vulnerability detection, outperforming both static-only and dynamic-only approaches, and significantly outperforming manual penetration testing in both recall and efficiency. The automated remediation pipeline further extends this capability by reducing average Time-to-Remediate from 72 hours to 4.5 minutes for supported vulnerability classes.

The system is particularly well-suited to the current Web3 development environment, where the adoption of AI coding assistants has accelerated the introduction of infrastructure vulnerabilities that traditional smart contract audits cannot detect. Veritas RedTeam provides the autonomous security counterpart to autonomous development, ensuring that the speed of code generation does not outpace the speed of security validation.

12. References

[1] Perry, N., Srivastava, M., Kumar, D., and Boneh, D. “Do Users Write More Insecure Code with AI Assistants?” Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS ‘24). Association for Computing Machinery, New York, NY, USA, 2024. DOI: 10.$^{1145}/_{3576915}$.3623177

[2] Rekt Database. “DeFi Hack Statistics 2020-2025.” https://rekt.news/leaderboard/

[3] CISA. “Known Exploited Vulnerabilities Catalog.” https://www.cisa.gov/known-exploited-vulnerabilities-catalog

[4] OWASP. “OWASP API Security Top 10.” https://owasp.org/www-project-api-security/

[5] Mitre Corporation. “Common Vulnerabilities and Exposures (CVE).” https://cve.mitre.org/

[6] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. “ReAct: Synergizing Reasoning and Acting in Language Models.” International Conference on Learning Representations (ICLR), 2023. https://arxiv.org/abs/2210.03629

[7] Neo4j. “Graph Database Platform.” https://neo4j.com/

[8] OpenVAS / GVM. “Greenbone Vulnerability Management.” https://www.greenbone.net/

[9] TruffleHog. “Secret Scanning Tool.” https://github.com/trufflesecurity/trufflehog

[10] Nuclei. “Fast and Customizable Vulnerability Scanner.” https://github.com/projectdiscovery/nuclei

[11] Veritas Protocol. “Smart Contract Audit Technical Report.” https://www.veritasprotocol.com/

[12] Veritas Protocol. “Veritas Explorer: Blockchain Forensics and Real-Time Monitoring.” https://explorer.veritasprotocol.com/