



RESEARCH-DRIVEN SMART CONTRACT SECURITY / WHITE PAPER V1.10

AI-Powered Smart Contract Security Intelligence

From Vulnerability Detection to Continuous Assurance for On-Chain Finance

| | MANDO SCAN Research & Product Team | | October 2026 | | Public Technical White Paper

About this white paper. MANDO SCAN combines **vulnerability detection** with **AI-assisted fix suggestions** to support smart-contract security review. This document presents the platform's scientific foundations, technical approach, published research evidence, and development roadmap. Research benchmarks apply to the datasets and experimental settings reported in the cited papers; they do not constitute performance guarantees for any specific deployment. This document is not an independent security audit, compliance certification, financial prospectus, or token offering.

Executive Summary

Smart contracts have become programmable infrastructure for decentralized applications, on-chain finance, tokenized assets, and increasingly complex institutional workflows. Their ability to execute transparent, deterministic rules creates a new security challenge: an implementation mistake can become a persistent, publicly accessible attack surface, and a conventional point-in-time review cannot fully cover how code changes, dependencies, configurations, and operational conditions evolve.

MANDO SCAN is an AI-powered smart-contract security platform offering **vulnerability detection and AI-assisted fix suggestions**, built on a sustained research program in smart-contract vulnerability analysis. Its central premise is that understanding smart-contract risk requires more than inspecting isolated tokens or matching predefined vulnerable patterns. Security analysis must connect **what code says** to **how code can execute**, **which functions interact**, and **where a suspected weakness occurs**.

MANDO's peer-reviewed research introduces a family of models that represent Solidity smart contracts as **heterogeneous contract graphs**, joining control-flow graphs and function-call graphs. These representations allow machine-learning models to distinguish different node and relation types and to make predictions at both the **contract level** and the **line level**. The research evolved from multi-metapath graph attention (MANDO), through a developer-facing prototype (MANDO-GURU), to heterogeneous graph transformers (MANDO-HGT), and to a hybrid of graph transformers and language-model-derived code representations (MANDO-LLM). [R1–R4]

MANDO SCAN combines **fine-grained vulnerability detection**, **contextual issue explanations**, and **AI-assisted fix suggestions** in a unified developer workflow. Planned enhancements include simulation-based fix validation, enterprise-specific customization, deeper development integrations, and continuous operational assurance.

The objective is not to replace expert security engineers or claim that a scan makes a contract safe. It is to help organizations **find important issues earlier**, **understand them faster**, **prioritize them consistently**, and **verify fixes with greater discipline**.

The MANDO SCAN thesis: *Understand the structure. Interpret the semantics. Explain the risk. Validate the fix. Continuously improve assurance.*

1. Vision: A Security Intelligence Layer for Programmable Value

1.1 The changing role of smart contracts

Smart contracts increasingly govern custody, exchange, lending, payments, governance, token issuance, asset servicing, and settlement logic. A failure may affect not only one application, but also upstream contracts, libraries, governance authorities, bridges, or external services. This makes security a systems problem as well as a code problem.

Traditional development often treats security as a checkpoint between implementation and deployment. That model is incomplete for software whose execution can be observed and triggered by arbitrary participants and whose dependencies and privileged controls may change after deployment.

MANDO SCAN's long-term vision is a **trust layer and automated security-analysis infrastructure** that fits into the full software and smart-contract lifecycle—from initial design through coding, release, and operational monitoring. The

platform delivers fine-grained vulnerability detection **and AI-assisted fix suggestions**. Additional lifecycle components follow an incremental roadmap; not all envisioned modules are commercially available today.

1.2 Design principles

1. **Structure-aware intelligence.** Analyze execution paths, branching, and function interactions rather than only surface syntax.
2. **Semantics-aware intelligence.** Combine structural graphs with code representations that help distinguish functionally different contexts.
3. **Actionable localization.** Trace predictions back to relevant lines, functions, and graph neighborhoods.
4. **Evidence before automation.** A finding should be reviewable, reproducible where possible, and distinguish prediction from verified exploitability.
5. **Human-governed remediation.** Treat suggested changes as candidates requiring tests, review, and approval.
6. **Workflow-native security.** Bring checks into the points where developers already build, review, and deploy software.
7. **Transparent uncertainty.** Report the limits of the model, the analysis surface, and the strength of supporting evidence.
8. **Research-to-product accountability.** Separate peer-reviewed experiments from production-level service claims.

2. The Problem: Why Smart Contract Security Remains Difficult

2.1 Security is contextual, not merely syntactic

The statement `call{value: amount}("")` is not necessarily vulnerable on its own. Its risk depends on surrounding conditions: when balances are updated, which external contracts may execute callbacks, how authorization is established, and whether the program preserves intended invariants. Similar reasoning applies to authorization gaps, unchecked return values, and transaction-ordering exposure.

A scanner that only matches keywords may generate alerts that are hard to act on, while a language-only model may miss long-range call relationships. Manual review can provide deeper understanding but takes expertise and time, and one review cannot automatically follow every subsequent change. No single analysis approach reliably resolves all these trade-offs.

2.2 A layered attack surface

Layer	Examples of risk	What security analysis needs to examine
Contract logic	Reentrancy, incorrect state transitions, authorization failures	Execution paths, state updates, privileged calls
Numerical and temporal behavior	Arithmetic issues, timestamps, order-dependent outcomes	Assumptions, constraints, transaction ordering
Calls and dependencies	Unchecked calls, interactions with external contracts	Call graph, error handling, trust boundaries
Protocol economics	Oracle assumptions, price manipulation, liquidity and collateral logic	Cross-component invariants and adversarial scenarios
Governance and upgrades	Overpowered roles, proxy configuration, unsafe changes	Roles, change controls, implementation relationships
Delivery and operations	Misconfiguration, unreviewed release, incident response	Review gates, regression testing, change history

The first three rows are closest to the published MANDO detection research. The latter layers motivate future complementary testing and assurance capabilities; they should **not** be interpreted as a claim that the current model detects all ecosystem or economic exploits.

2.3 Why complementary methods matter

Security professionals rely on static analyzers, symbolic execution, fuzzing, formal methods, manual auditing, monitoring, and increasingly machine learning. Each has strengths: formal specifications can check precise invariants; tests can demonstrate concrete behaviors; static analysis can scale; expert review can consider protocol intent. MANDO SCAN is designed to **complement**, not invalidate, these techniques.

3. Scientific Foundation: The MANDO Research Lineage

MANDO SCAN's differentiation is rooted in a succession of research contributions rather than a generic prompt applied to Solidity.

Research stage	Core contribution	Relevance to the platform
MANDO (2022) [R1]	Multi-level, heterogeneous graph embeddings and metapath-aware attention over fused control-flow and call graphs	Structure-aware, contract- and line-level detection
MANDO-GURU (2022) [R2]	A developer-facing research tool with Solidity scanning, vulnerability visualization, and line-level inspection	Explainable analyst/developer experience
MANDO-HGT (2023) [R3]	Heterogeneous graph transformers and relation-aware representations for source code and bytecode research	More expressive structural learning; research support for source or bytecode analysis
MANDO-LLM (2026) [R4]	Compiler-independent syntax-based graphs, dynamic meta relations, and LLM-derived code features combined with graph transformers	Joint structural and semantic code understanding

MANDO, MANDO-GURU, MANDO-HGT, and MANDO-LLM describe the successive research contributions. **MANDO SCAN** applies this research foundation in a product platform offering vulnerability detection and fix suggestions. Capabilities demonstrated in research and product release scope are evaluated separately.

4. Technical Architecture

4.1 A layered intelligence architecture

Figure 1 combines the **end-to-end analysis flow** with the **research-to-product layering**. Rather than treating parsing, graph construction, semantic encoding, and remediation as one undifferentiated model, it shows where complementary representations are constructed and subsequently joined. The upper portion is a conceptual synthesis of published MANDO methods; the lower portion identifies MANDO SCAN's available reporting and AI-assisted fix suggestions. It is not a source-code-level disclosure of the commercial implementation. Simulation-based validation, expanded development-workflow integrations, and continuous operational assurance are roadmap directions.

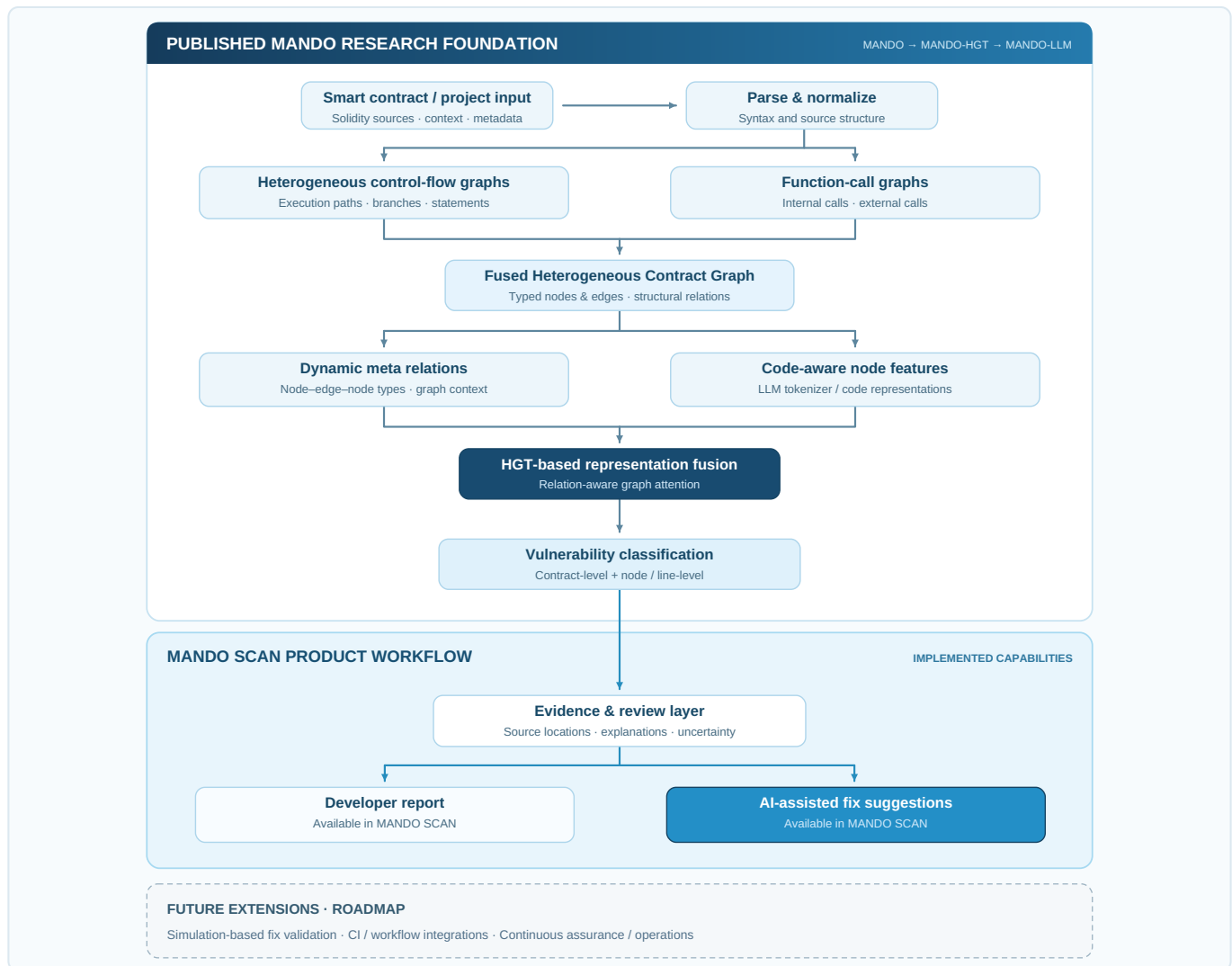


Figure 1. Combined MANDO research and MANDO SCAN workflow: input → CFG/call-graph construction → heterogeneous graph fusion → parallel structural and code-aware representations → HGT → vulnerability classification → evidence and implemented developer reporting/fix suggestions. The core research steps are adapted from [R1, R3, R4]; product-layer reporting and fix suggestions are not presented as experimentally evaluated by those papers. The separately styled roadmap strip is not part of the implemented scan.

4.2 Input, parsing, and normalization

For source-code analysis, a user or integration supplies Solidity code and, where needed, relevant project context. The published MANDO-LLM method uses a syntax-based graph generator supported by **tree-sitter**, which can create an abstract syntax tree without depending on a complete contract-compilation environment. This was introduced to reduce friction from missing libraries and compiler-version incompatibilities found in earlier workflows. [R4]

Project-level metadata can support this workflow when supplied by an integration; the available inputs depend on the deployment configuration.

Source versus bytecode matters. MANDO-HGT investigated source and bytecode as research inputs. MANDO-LLM’s described architecture and evaluation focus on Solidity source code. Support for arbitrary deployed bytecode, verified source retrieval, other virtual machines, and cross-chain semantics must be validated independently in a commercial release. [R3, R4]

4.3 Building the heterogeneous contract graph

A conventional **control-flow graph (CFG)** captures possible flows of execution within functions. A **call graph (CG)** captures invocations between functions or contracts. These representations are useful separately but omit relationships when considered in isolation.

MANDO constructs a **heterogeneous contract graph (HCG)** by combining them and preserving differences among node and edge types. Examples of node types include contract declarations, function definitions, expressions, branches, return statements, loop-related nodes, and fallbacks. Examples of relations include sequential flow, conditional branches, internal calls, and external calls. [R1, R3, R4]

Formally, the research represents an HCG as a directed heterogeneous graph:

$$G = (V, E, \tau, \varphi)$$

$$\tau: V \rightarrow A \quad \cdot \quad \varphi: E \rightarrow R$$

where **V** and **E** are nodes and edges; **A** is the set of node types; **R** is the set of edge types; and **τ**, **φ** map each node and edge to its respective type. [R4]

This representation matters because an **IF** branch leading to an external call carries meaning distinct from an ordinary sequential transition, even if both can be drawn as arrows in a simplified graph.

Figure 2 illustrates this relationship through a simplified reentrancy-prone withdrawal: the control-flow graph follows the external call and subsequent balance update, while the function-call graph reveals the possible callback path. Fusing these typed relations helps the model analyze the risk in context.

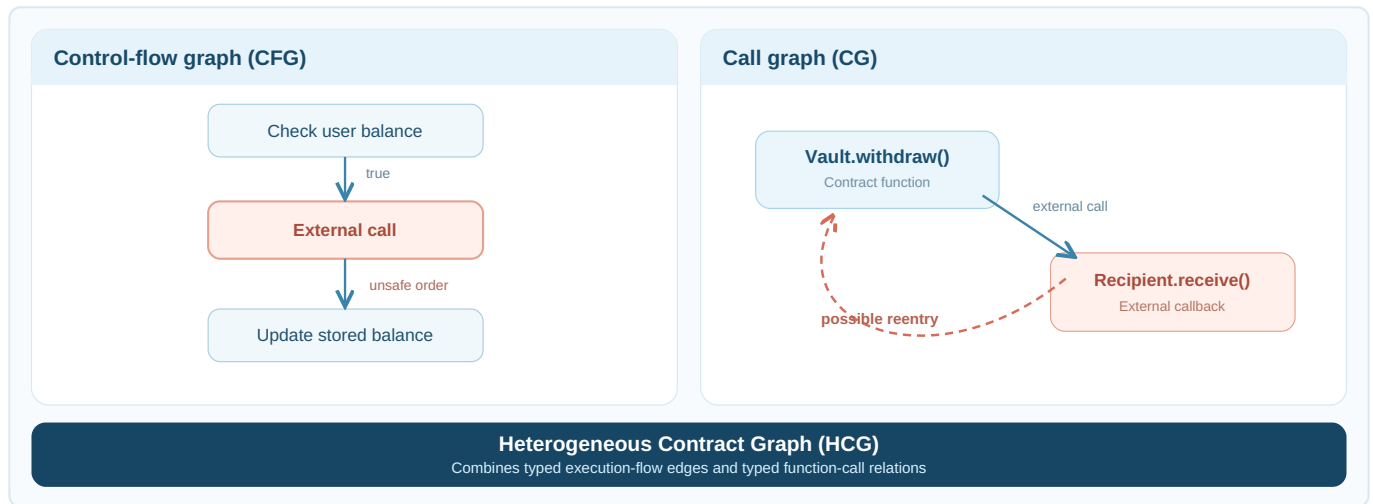


Figure 2. Illustrative CFG/CG fusion inspired by the reentrancy case studies in MANDO (DSAA 2022) and MANDO-LLM (TOSEM 2026) [R1, R4]. This simplified redraw shows the analysis principle, not a scan result or a complete execution trace.

4.4 Dynamic meta relations

A **meta relation** captures the semantic type of one edge and its endpoints:

$$\rho(e = (s, t)) = \langle \tau(s), \varphi(e), \tau(t) \rangle$$

For example, a relation of the form `<IF, true-branch, EXTERNAL_CALL>` differs from `<EXPRESSION, next, RETURN>`. The latter examples are illustrative; actual node and edge labels follow the published graph schema.

MANDO-LLM extracts relations dynamically from graph structures rather than requiring all relation patterns to be fixed by hand. The paper discusses up to **18 node types and five edge types**, with only relevant combinations instantiated in practice. [R4] This is a research property of the described method, **not** a promise of zero configuration or universal compatibility with future languages.

4.5 Heterogeneous Graph Transformers

A **heterogeneous graph transformer (HGT)** learns representations of graph nodes while accounting for the types of connected nodes and their relationships. Attention mechanisms can assign different importance to different neighbor-

hoods. Graph-level representations allow contract-level classification, while node-level representations support more localized predictions. [R3, R4]

Unlike a flat sequence classifier, this design can propagate information across relevant code structures. Importantly, an attention weight or highlighted subgraph is **evidence about model behavior**, not a formal proof that the model's conclusion is correct.

4.6 LLM-derived code semantics

MANDO-LLM extends graph topology with language-model-derived representations of code associated with graph nodes. A code-focused tokenizer or encoder supplies features that capture information present in source snippets, while the graph transformer contributes relationships that would be difficult to infer reliably from isolated snippets. [R4]

The combination is essential: **the model is not simply asking a general-purpose chatbot to decide whether a contract is secure**. The research investigates how semantic code features and typed graph structure interact in a learned vulnerability detector.

An implementation should keep a strict boundary between **detection predictions, natural-language explanations, and proposed code changes**. All three have different levels of reliability and require different validation.

4.7 Multi-granularity classification and localization

The research uses a two-phase vulnerability detection approach to predict vulnerability categories at contract granularity and identify relevant vulnerable nodes or lines. [R4] The result can support an analyst workflow with a high-level risk summary, followed by code-level inspection.

MANDO SCAN's implemented user workflow presents vulnerability findings and associated fix suggestions. The broader reference reporting model can add review, validation and provenance fields as features evolve. **Exact exploitability, severity, and remediation correctness cannot be inferred solely from a model label**.

5. Detection Scope and Security Taxonomy

5.1 Seven vulnerability families examined in the research

The MANDO-LLM evaluation concentrates on seven vulnerability categories shared across its benchmark sources. [R4]

Category	Representative concern	What a developer needs to verify
Access control	A sensitive operation is callable by an unauthorized actor	Role checks, call path, permissions, upgrade authority
Arithmetic	Numerical logic produces unintended or unsafe behavior	Boundaries, compiler semantics, precision and units
Denial of service	Conditions prevent important functions from completing	Failure paths, external dependencies, gas and loops
Front running	Transaction ordering can disadvantage intended users	Timing, exposure, transaction-level assumptions
Reentrancy	External execution re-enters before critical state is finalized	Call ordering, invariant maintenance, callbacks
Time manipulation	Logic trusts controllable or imperfect timing inputs	Timestamp assumptions and tolerance
Unchecked low-level calls	A low-level call result is ignored or incorrectly handled	Return-value checks, revert behavior, error recovery

These categories are **the evaluation scope of published models**, not a representation that every variant of each weakness is detectable or exploitable.

5.2 Beyond classical vulnerabilities: insecure practices

Some dangerous practices are not immediate exploits. Examples include poorly documented privileged authority, non-standard error-handling conventions, unsafe assumptions about external contracts, weak validation boundaries, and fragile upgrade procedures.

MANDO SCAN brings a practical remediation step into the security review process: a detected issue can be accompanied by **contextual explanations and fix suggestions**. This supports developer understanding and review without treating every flagged coding practice as an exploitable vulnerability. Suggested fixes still require testing and approval. Broader policy-aware guidance and knowledge-grounded reporting remain possible enhancements.

5.3 Standards-aware reporting

Where relevant, findings may be cross-referenced to established smart-contract weakness taxonomies such as the **Smart Contract Weakness Classification (SWC)** and comparable security-control or coding-guideline frameworks. The existence of such mapping does **not** imply certification under a specific standard or that automated scanning alone establishes regulatory compliance.

The practical goal is **standards-adaptable reporting**: an enterprise can associate security findings with its internal policies, acceptable risk thresholds, required reviewer roles, and documented development controls. This remains a configurable workflow ambition until implemented and independently assessed for a given environment.

6. Explainable Findings and Responsible Remediation

6.1 What an actionable finding should contain

A useful security finding answers more than “is this suspicious?” MANDO SCAN **supplies fix suggestions alongside vulnerability findings**. A more comprehensive reference record for existing and evolving workflows may include:

1. **Issue identity**: type, title, scan version, source artifact and affected location.
2. **Technical evidence**: suspicious code path, relevant graph neighborhood, and implicated state transitions.
3. **Interpretation**: why the pattern could violate a security invariant, and under what assumptions.
4. **Confidence and triage**: model confidence where calibrated, rule/test corroboration, review status, and an impact estimate assigned through a separate risk process.
5. **Remediation candidate**: a code or design change with rationale and identified limitations.
6. **Validation outcome**: tests performed, reproducible evidence, reviewer approval, and any unresolved residual risk.

MANDO SCAN provides **vulnerability findings and associated fix suggestions**. The additional elements above form a **reference reporting schema** for security review workflows; exploit validation, reviewer approval, and other evidence fields require their own verification processes.

6.2 Illustrative example: a reentrancy risk

Consider a simplified withdrawal sequence:

```
// Illustrative anti-pattern: external interaction precedes state update.
(bool sent, ) = msg.sender.call{value: amount}("");
require(sent, "Transfer failed");
balances[msg.sender] -= amount;
```

The security concern is not the `call` instruction by itself. It is that an external recipient may execute code before the contract updates the user's balance. A graph-aware analyzer can consider the branch, external call, and subsequent state update together. This closely reflects the motivation for MANDO's original graph representations. [R1, R4]

A suitable **fix suggestion** for this illustrative pattern may be to apply the *checks—effects—interactions* principle, update state before making the external call, and consider appropriate reentrancy protections. A responsible workflow must then test correctness across the full withdrawal function, accounting for failed transfers, accounting invariants, and interactions with other callable methods. The three-line fragment alone is not a complete vulnerability proof or a production-ready patch.

6.3 Why a language model should not self-certify its fixes

An LLM-assisted system may suggest plausible code that changes token behavior, breaks compatibility, introduces a different vulnerability, or silently assumes a nonexistent invariant. **MANDO SCAN provides fix suggestions**; these are **recommendations**, not automatically verified patches. For safe adoption, developers should inspect suggested changes, review diffs, test intended behavior, and approve any modification before deployment.

The supported and planned stages can be represented as:

```
Detect [implemented] → Explain / Suggest Fix [implemented] → Developer Review → Simulate / Test [planned  
automated environment] → Approve → Re-scan .
```

MANDO SCAN provides **AI-assisted fix suggestions** as part of its vulnerability detection workflow. Automated simulation-based testing and fix validation are separate development priorities.

7. AI-Assisted Fix Suggestions and Knowledge Enhancement

7.1 Remediation assistance

MANDO SCAN provides AI-assisted fix suggestions alongside vulnerability detection. The platform connects security findings with contextual explanations and recommended changes, helping developers understand potential weaknesses and evaluate practical remediation options.

The goal is to make findings actionable without overclaiming automated security. Suggested fixes are **reviewable candidates**, not proof that an issue is resolved or that intended business logic is preserved. Developers remain responsible for examining changes and validating them with relevant tests.

7.2 Future knowledge-grounded capabilities

A longer-term opportunity is a curated, versioned knowledge layer that can enrich recommendations with project-specific coding conventions, verified weakness descriptions, relevant standards, historical remediation examples, and applicable enterprise policies. Such a capability would require careful source attribution, retrieval quality assessment, safeguards against untrusted content, and explicit controls for confidential code.

Related research has also investigated LLM-assisted identification of smart-contract coding practices and retrieval-supported audit reasoning [R5].

These capabilities are **future enhancements**, separate from the platform's existing vulnerability detection and fix suggestions.

7.3 Evaluating remediation quality

Fix-suggestion effectiveness should be measured separately from vulnerability detection. Important measures include **patch correctness, preservation of intended behavior, regressions, newly introduced weaknesses, developer acceptance, reproducibility, and review time saved**. The MANDO research papers report detection results, not a universal patch-success rate for the deployed product.

8. Evaluation: What the Research Shows—and Does Not Show

8.1 Published research evidence

The following results are from **different papers, datasets, tasks, classifiers, and comparison baselines**. They are deliberately not collapsed into a universal commercial accuracy figure.

Study	Evaluation context	Absolute F1 scores (Buggy-F1) reported in the research
MANDO (2022) [R1]	Solidity source code, seven vulnerability categories; annotated vulnerable contracts	81.108%–90.51% line-level F1 across the seven categories. Examples: 90.51% for Reentrancy and 90.29% for Time Manipulation (DSAA Table III).
MANDO-GURU (2022) [R2]	Developer-facing tool based on the MANDO graph-learning method; SmartBugs and SolidiFI evaluations	89.86% contract-level F1 for Front Running and 90.51% line-level F1 for Reentrancy in the underlying MANDO model [R1, Tables II–III]. The MANDO-GURU tool paper [R2] does not report a separate absolute-F1 table.
MANDO-HGT (2023) [R3]	Source-code contract and line classification; seven vulnerability categories; one-hot node-feature configuration	80.11%–95.68% contract-level F1 (MSR Table II, <i>source code</i> results) and 86.81%–95.04% line-level F1 (Table III). Examples: 92.59% for line-level Reentrancy and 95.04% for line-level Time Manipulation. Other node-feature configurations report different scores.
MANDO-LLM (2026) [R4]	Solidity source code, seven vulnerability categories; Dataset A; best Buggy-F1 per category among reported tokenizer variants	86.65%–96.18% contract-level F1 (TOSEM Table 3) and 95.42%–98.23% line-level F1 (Table 5). Examples: 96.78% for line-level Reentrancy and 98.23% for line-level Denial of Service. Dataset B reports different results.

Reading the table. All percentages refer to the positive (“buggy”) class’s **F1-score**, not classification accuracy, and are specific to the indicated task and experimental configuration. Ranges cover different vulnerability types—not a confidence interval or a single overall score. The MANDO-GURU figures come from its underlying MANDO model [R1], not a separate tool-only benchmark. Their denominators, labels, class balance, data sources, and baseline definitions differ. In particular, the 2026 study describes balanced contract-level mixtures for evaluation, and line-level results on buggy contracts; neither setup alone establishes performance on an arbitrary real-world prevalence of security issues. [R4]

8.2 Interpreting precision, recall, and F1

Precision is the fraction of flagged items that are true positives. Recall is the fraction of actual positive items that are detected. F1 is the harmonic mean:

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{F1 score} = 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$$

For imbalanced security datasets, both **positive-class F1** and the behavior of false alarms matter. A score on one benchmark cannot be substituted for production false-positive rate, exploit-detection probability, financial loss reduction, or audit completion time.

8.3 Benchmarks for a commercial platform

Published research results and release-specific performance are distinct. A reproducible product evaluation can measure:

- **Detection:** per-category precision, recall, F1, false positives, false negatives, and classification calibration.
- **Localization:** whether reported source spans overlap reviewer-confirmed root causes.
- **Robustness:** unseen projects, Solidity versions, dependency changes, compiler settings, and previously unseen patterns.
- **Runtime:** latency per contract, workload size, throughput, resource usage, and infrastructure cost.
- **Remediation:** human acceptance rate, fix correctness, regression rate, and whether independent tests confirm resolution.
- **Deployment:** multi-tenant isolation, data protection, audit logs, and integration reliability.
- **Human factors:** developer time saved, review burden, comprehensibility, and missed critical findings.

Fast turnaround and low false-alarm rates are engineering objectives. These objectives do not constitute production performance guarantees or service-level commitments.

9. A Product Model for the Full Smart Contract Lifecycle

MANDO SCAN's platform strategy is to connect detection to decision-making and delivery. The product vision can be organized into six stages.

Lifecycle stage	Security need	MANDO SCAN direction	Status characterization
Design	Understand assets, actors, assumptions, and invariants	Requirement-aware threat modeling	Longer-term direction
Coding	Identify suspicious patterns before merge or release	AI-powered fine-grained vulnerability detection	Implemented in MANDO SCAN; research-supported detection methods
Remediation	Explain a finding and provide possible corrections	AI-assisted fix suggestions linked to vulnerability findings	Implemented in MANDO SCAN; developer review still required
Pre-deployment	Check exploit scenarios and validate changed code	Simulation-based testing and regression checks	Planned development feature
Integration	Fit into approval, development and enterprise workflows	Custom integration, reports, policy mapping and APIs	Planned/customizable deployment path
Operations	Track deployed changes and evolving exposure	Monitoring, investigation and attack-response support	Longer-term direction

MANDO SCAN's **vulnerability detection** and **AI-assisted fix suggestions** support the coding and remediation stages today. The broader development strategy extends these capabilities toward simulation-based validation, enterprise workflow integration, and operational monitoring. Availability of specific integrations and deployment options depends on the applicable product configuration.

10. Deployment and Enterprise Workflows

10.1 Developers and small teams

For an individual developer, the intended value proposition is a fast feedback loop: submit or connect a contract, inspect security findings, understand affected code, and take corrective action before release. The relevant measure of success is **useful feedback per unit of developer effort**, not simply the number of warnings emitted.

10.2 Blockchain-native organizations

Protocol teams often modify contracts in coordination with token interfaces, external services, multisig controls, and other on-chain components. Their security needs include regression tracking, dependency-aware review, and reproducible analysis across versions. MANDO SCAN's envisioned workflow integration can support repeated checks as part of a layered audit program.

10.3 Financial institutions and enterprises

Banks, asset managers, payment firms, and enterprises exploring tokenized assets or programmable settlement require more than a vulnerability label. They may need documented approvals, reproducible evidence, separation of duties, and deployment choices that protect confidential code. MANDO SCAN focuses on developers, blockchain-native organizations, and institutions exploring tokenization, digital assets, and programmable finance.

A **private deployment, enterprise-specific knowledge policy, or on-premises workflow** may be appropriate in some environments, but should be described as an offering only when supported by a specific product configuration and contract. This white paper does not promise certified compliance, data residency, or an enterprise SLA.

10.4 An illustrative integration sequence

Source change → Automated analysis → Developer triage → Security review → Fix and testing → Approval → Deployment evidence .

This is an **integration pattern**, not a representation that named API endpoints, repository plugins, or CI connectors already exist. Implementations should preserve an immutable relationship among the reviewed source revision, analysis/model version, reviewer decision, and delivered artifact.

11. Security, Privacy, and Trustworthiness of the Analyzer

A security scanner itself belongs inside an organization's threat model. MANDO SCAN therefore treats the following as **deployment requirements and engineering priorities** rather than undocumented service guarantees.

11.1 Confidential code and customer control

Enterprise deployments should define ownership, retention, training-use restrictions, access control, deletion processes, and whether source code is processed outside the customer's approved environment. No confidentiality or non-training guarantee should be inferred from this white paper alone; those assurances require implemented controls and applicable agreements.

11.2 Adversarial and misleading inputs

Contracts may include adversarial naming, deceptive comments, generated source, deeply nested structures, or compiler-sensitive constructs. AI-assisted explanation also creates prompt-injection risks. A safe design validates parsers, limits resource consumption, isolates untrusted code, and prevents source content from issuing operational instructions to the analyzer.

11.3 Versioning and reproducibility

A meaningful scan report should identify the analyzed source revision, available compiler context, detector version, rule/knowledge-base version, timestamp, and whether dynamic validation was actually performed. A later model or contract change can invalidate the conclusions of an older scan.

11.4 Responsible disclosure

New findings in third-party contracts should follow coordinated disclosure practices where appropriate. A scanner finding is not authorization to probe live systems, transfer funds, publish exploit details, or bypass an owner's access controls.

12. Roadmap and Milestone Logic

MANDO SCAN builds on its existing **vulnerability detection and AI-assisted fix suggestions** to expand secure remediation, testing, enterprise integration, and continuous assurance. The following roadmap reflects the platform's development priorities.

Phase	Product emphasis	Evidence of meaningful completion
Implemented foundation	Fine-grained vulnerability detection and AI-assisted fix suggestions in the MANDO SCAN platform	Existing product capability; performance claims require separately published release-specific evidence
Remediation hardening	Systematic review, safer change suggestions, and fix-quality measurement	Reproducible reviewer-accepted changes; quantified patch correctness and regression outcomes
Validated assurance	Simulation, regression testing, release-gate integration	Demonstrated reproduction/verification on agreed test suites
Enterprise workflows	Custom integrations, policy mapping, reporting, configurable deployment	Successful customer-specific pilots against documented requirements
Continuous intelligence	Broader ecosystem visibility, ongoing reviews, operational signals, additional languages	Independently evaluated capabilities and clear scope limitations

Roadmap milestones are planned capabilities, not statements of completed commercial deployments or contractual commitments. Product releases and customer-specific offerings are documented separately.

13. Ecosystem Positioning and Adoption

MANDO SCAN is built for a practical place in the security stack: **an intelligent, repeatable analysis layer between everyday software development and independent assurance activities**. It can assist security engineers and developers in deciding what to inspect and what to test next. It does not purport to replace third-party audits, invariant-based testing, formal verification, penetration testing, or incident-response preparedness.

Its potential differentiators are:

- **Research continuity:** a peer-reviewed lineage that specifically addresses Solidity contract vulnerability detection. [R1–R4]
- **Typed graph understanding:** control-flow and function-call representations learned as heterogeneous relationships. [R1, R3, R4]
- **Fine-grained feedback:** outputs intended to direct attention toward relevant lines and functions rather than only a document-level warning. [R1, R2, R4]
- **Hybrid AI approach:** learned graph relationships augmented with code-semantic representations, rather than an unstructured standalone chat prompt. [R4]

- **Practical remediation assistance:** implemented fix suggestions alongside vulnerability detection; with testing, organizational review, and broader lifecycle integration continuing as separate assurance steps.

These are **design and research differentiators**, not an unsupported blanket superiority claim against every audit provider or tooling vendor.

14. Risk Boundaries, Limitations, and Open Questions

The platform's effectiveness is constrained by data quality, class imbalance, unseen patterns, model calibration, parser coverage, deployment configuration, and the difference between a **potential weakness** and an **exploitable vulnerability**.

Specific limitations include:

1. **False positives and false negatives cannot be eliminated.** Absence of an alert is not evidence of absence of risk.
2. **Dataset labels are imperfect.** Public benchmarks contain disputed classifications, synthetic cases, and different sampling distributions.
3. **Source-code coverage does not automatically imply bytecode or chain coverage.** Chain-specific semantics need separate validation.
4. **Business-logic vulnerabilities may require external specifications.** Economic exploits, bridge risk, price assumptions, and off-chain controls often cannot be assessed from one contract alone.
5. **LLM explanations can hallucinate.** They require grounding, review, and confirmation by tools or domain experts.
6. **A localized graph node is not always the root cause.** Interpretation can be useful without being causal proof.
7. **Simulation is not proof of correctness.** A passing test suite covers only its tested scenarios and assumptions.
8. **Security certification is a separate process.** A generated report is not an independent audit opinion or regulatory attestation.

These limits motivate the broader assurance workflow. The responsible goal is to improve the quality and timeliness of security decisions—not to declare any contract invulnerable.

15. Scope: Protocol and Token Considerations

MANDO SCAN is a smart-contract security platform, not a blockchain consensus protocol or cryptocurrency. This white paper does not offer or describe a native token, token sale, staking mechanism, or investment rights.

No native token or token offering is necessary for the platform's smart-contract security capabilities. Any future token-related initiative would be documented separately.

16. Conclusion

Programmable finance needs security tools that understand not only lines of source code but also the structural relationships that give those lines their meaning. MANDO's research addresses this requirement through heterogeneous control-flow and call representations, graph-based learning, and a subsequent integration of language-model-derived code semantics. [R1–R4]

MANDO SCAN provides **fine-grained vulnerability detection and AI-assisted fix suggestions**, with future development focused on **test-supported validation**, enterprise integration, and **continuous security assurance**. This progression connects published research to practical security workflows while maintaining a clear distinction between available capabilities and future development.

The long-term opportunity is a security intelligence layer that helps developers, security professionals, and institutions build and operate more trustworthy on-chain systems—while remaining explicit that no automated tool can guarantee their safety.

MANDO SCAN — See the Risk. Build With Confidence.

Website: mandoscan.com

Appendix A. Illustrative Finding Record

The following example illustrates a **proposed report format**, not a response captured from a commercial MANDO SCAN installation.

```
finding_id: "illustrative-001"
category: "Reentrancy"
asset: "ExampleVault.sol"
location:
  function: "withdraw(uint256)"
  relevant_lines: "example only"
evidence:
  - "external call occurs before a balance update"
  - "control-flow path and call relationship require review"
assessment:
  classification: "potential vulnerability"
  exploitability: "not established by prediction alone"
  review_status: "human review required"
recommendation:
  - "review checks-effects-interactions ordering"
  - "consider appropriate reentrancy defenses"
  - "test withdrawals, callbacks, and failure paths"
validation:
  simulation_performed: false
  independent_audit: false
```

Appendix B. Glossary

Term	Meaning
AST	Abstract Syntax Tree; structural representation produced by parsing code
CFG	Control-Flow Graph; possible execution-flow relationships
CG	Call Graph; relationships between function invocations
HCG	Heterogeneous Contract Graph; typed combined representation of contract relationships
GNN	Graph Neural Network; learned model for graph-structured information
HGT	Heterogeneous Graph Transformer; relation-aware graph attention architecture
LLM	Large Language Model; here used for code representation and potentially for explanations
RAG	

Term	Meaning
	Retrieval-Augmented Generation; response generation grounded in retrieved source material
Metapath / meta relation	Typed relational patterns between or across graph nodes
F1 score	Harmonic mean of precision and recall on a specified classification task
SWC	Smart Contract Weakness Classification taxonomy
POC / MVP	Proof of Concept / Minimum Viable Product
SAST	Static Application Security Testing
False positive / negative	Incorrect alert / missed issue

References and Provenance

Research publications

- [R1] H. H. Nguyen, N.-M. Nguyen, C. Xie, Z. Ahmadi, D. Kudendo, T.-N. Doan, and L. Jiang. “**MANDO: Multi-Level Heterogeneous Graph Embeddings for Fine-Grained Detection of Smart Contract Vulnerabilities.**” *Proceedings of the 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA 2022)*, Shenzhen, China, 2022, pp. 1–10. [DOI: 10.1109/DSAA54385.2022.10032337](https://doi.org/10.1109/DSAA54385.2022.10032337).
- [R2] H. H. Nguyen, N.-M. Nguyen, H.-P. Doan, Z. Ahmadi, T.-N. Doan, and L. Jiang. “**MANDO-GURU: Vulnerability Detection for Smart Contract Source Code by Heterogeneous Graph Embeddings.**” *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*, Singapore, 2022, pp. 1736–1740. [DOI: 10.1145/3540250.3558927](https://doi.org/10.1145/3540250.3558927).
- [R3] H. H. Nguyen, N.-M. Nguyen, C. Xie, Z. Ahmadi, D. Kudendo, T.-N. Doan, and L. Jiang. “**MANDO-HGT: Heterogeneous Graph Transformers for Smart Contract Vulnerability Detection.**” *Proceedings of the 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR 2023)*, Melbourne, Australia, 2023, pp. 334–346. [DOI: 10.1109/MSR59073.2023.00052](https://doi.org/10.1109/MSR59073.2023.00052).
- [R4] N.-M. Nguyen, H. H. Nguyen, T. L. Le, Z. Ahmadi, T.-N. Doan, D. Wu, and L. Jiang. “**MANDO-LLM: Heterogeneous Graph Transformers with Large Language Models for Smart Contract Vulnerability Detection.**” *ACM Transactions on Software Engineering and Methodology*, 35(6), Article 144, May 2026. [DOI: 10.1145/3765751](https://doi.org/10.1145/3765751).
- [R5] Z. Li, X. Li, W. Li, and X. Wang. “**SCALM: Detecting Bad Practices in Smart Contracts Through LLMs.**” *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(1), 2025, pp. 470–477. [DOI: 10.1609/aaai.v39i1.32026](https://doi.org/10.1609/aaai.v39i1.32026). Independent related research.

Product status and roadmap. Implemented capabilities and planned directions reflect MANDO SCAN’s own product description as of October 2026. They are not independent benchmark certifications or contractual commitments.

Attribution. The scientific foundation of MANDO SCAN is documented in the **published peer-reviewed MANDO research papers** [R1–R4]. This white paper summarizes that research and MANDO SCAN’s platform direction without reproducing copyrighted article figures, datasets, or source code. Academic author profile: [Hoang H. Nguyen, ORCID 0000-0003-0611-4634](https://orcid.org/0000-0003-0611-4634). Research citations do not imply endorsement by associated institutions, conferences, or publishers.

Version: 1.10 · **Issued:** October 2026 · **Classification:** Public technical white paper · **Status:** Release candidate