

Toward general security validation

MagiChain Security · October 2026

Type: research direction and product scope. No measured results yet.**Scope:** EVM smart contracts, and payments that software agents initiate.

Most smart contract checks look for known bugs. We want a check that can evaluate a contract it has never seen against an explicit security requirement, and say how strong its evidence is. This overview sets out four lessons that shape that goal, each tied to a real incident; the three areas the product will validate; the six validators that produce the evidence; and how we will measure them. Price-oracle manipulation is the worked example throughout. Our companion note treats it in full [9].

1 What we set out to answer

Can a validator judge an unfamiliar contract against a stated security requirement, rather than against a list of past bugs, and report not only *what* it found but *how sure* it is?

Two lines of academic work make this plausible. Securify [1] showed that security properties can be written as compliance and violation patterns over a contract’s semantics, so one pattern covers many implementations. VerX [2] showed that temporal safety properties, which must hold across any sequence of transactions, can be verified automatically for real contracts. Our work asks how these foundations combine with faster, less exact methods in a product developers will actually run before they deploy.

2 Four lessons that shape the product

1. Whether code is safe depends on what its values are used for. Code can do exactly what it says and still lose funds. In October 2021, Cream Finance read a vault’s price per share correctly. The problem was that it used that number as collateral value, and anyone could inflate it by donating tokens to the vault; about \$130M was lost [9]. Reading the same number to display a balance would have been harmless. A validator therefore has to know what each value is *for*: which assets, permissions and external inputs give the logic its meaning.

2. One requirement covers many exploits, but only the ones it models. bZx (2020), Harvest (2020), Cream (2021) and Inverse (2022) had different code and different attack paths. All four broke one requirement: *a price that sets collateral or redemption value must not be movable by the attacker within the attack’s time horizon* [3,4,9]. Written once as a property, that requirement applies to code we have never seen. Its reach is still bounded by what it models. Mango Markets (2022) used an oracle that reported its market correctly; the market itself was manipulated, and a rule about price sources alone would not have flagged it [9].

3. A protective check counts only if it actually enforces the constraint. Inverse Finance had a time-weighted average price, the textbook defence, and was still exploited because of how the average was sampled [9]. A one-block average is a spot price under another name. A deviation check against a reference the attacker can also move checks nothing. A guard on one branch does not protect the other. Validators must report a guard’s parameters and whether it runs on every path, not merely that it exists.

4. Different methods support different conclusions. A matched signature says “this resembles something bad”. A profitable replay on a forked chain says “this is exploitable”. A proof says “this property holds under these assumptions”. Merging these into one score throws away the most useful information. The product keeps four kinds of verdict apart: a *confirmed issue*, a *warning*, a *guarantee* stated with its assumptions, and an *unresolved* risk.

3 What the product will validate

Area	Example requirement	Attacks it targets
Actions and asset movement	Only authorised roles can move funds or change ownership; no call transfers more than its caller is owed	Missing owner checks, <code>tx.origin</code> authentication, unsafe <code>delegatecall</code> , Ether leaks
External interactions	A value that sets collateral or redemption amounts can't be moved by the caller in the same transaction; state is updated before external calls	Price-oracle manipulation [9], reentrancy
State and execution sequences	Invariants hold after <i>any</i> sequence of transactions, e.g. total supply equals the sum of balances [2]	Multi-block manipulation, broken storage invariants, grieving across calls

4 Six validators, one verdict

Each validator fails in a different way, which is the point of running all six. Existing tools, benchmarked together, detected fewer than half of a set of annotated vulnerabilities [6].

Validator	What it contributes	Where it is blind	Evidence it yields
Classic signatures	Matches code cloned from exploited protocols, and known-bad deployers, in milliseconds	Anything new	Resemblance
Static analysis	Traces every flow from an input to a sensitive action and checks which guards dominate it [1,5]	Over-approximates; can't judge whether a guard's threshold is adequate	Possible issue
Formal verification	Proves a property on every path, or returns a counterexample [2]	Needs a written property and stated assumptions; costly on large code	Guarantee or counterexample
Dynamic analysis	Replays attacks, flash loans included, on a forked chain and measures the profit	Covers only the paths it exercises	Confirmed exploit
Deep learning model	Scores unseen variants, including bytecode with no source [8]	Opaque; only as good as its labels	Risk score
LLM code analysis	Reads what a value is for, triages other engines' findings and explains fixes [7]	Not deterministic; can be confidently wrong	Candidate issue, explanation

From evidence to a verdict. Cheap engines run first, the LLM triages and explains, and the expensive engines confirm. A finding becomes a confirmed issue only with a counterexample from the prover or a profitable replay from dynamic analysis. Otherwise it stays a warning, with its evidence attached. One engine's false positive should not block a deployment, and one engine's blind spot should not let one through.

5 Connected direction: agent payments

The same question arises when software agents pay. An x402 payment can be correctly signed, inside every spending limit, sent to an allowlisted merchant, and still not be what the user asked for. Our attack taxonomy calls this an in-policy but off-intent payment, and makes it the main research target. Our first finding showed that even the wallet-side spending policy underneath is easy to misconfigure. In both settings the gap is the same. A requirement that matters, whether the user's intent or a price that must not move, is not represented in what gets authorised. Our agent-payment work checks each payment against the intent mandate the user signed (AP2), using the same verdict model as above.

6 How we will measure it

- **Corpus.** Reproductions of real incidents, each paired with the protocol's patched version, plus unrelated audited contracts as controls [10]. Patched pairs are the fairest false-alarm test, since the code is nearly identical.
- **Measures.** Coverage per requirement; false alarms on patched versions; reproducibility, meaning the same verdict on repeated runs, which matters most for the LLM and the model; and cost per contract.
- **Reporting.** Rates across the corpus, never a single anecdote. The first requirement we will measure is the

oracle-manipulation rule from our companion note [9].

References

- [1] P. Tsankov et al. Securify: Practical security analysis of smart contracts. *ACM CCS*, 2018.
- [2] A. Permenev et al. VerX: Safety verification of smart contracts. *IEEE S&P*, 2020.
- [3] K. Qin, L. Zhou, B. Livshits, A. Gervais. Attacking the DeFi ecosystem with flash loans for fun and profit. *Financial Cryptography (FC)*, 2021.
- [4] L. Zhou et al. SoK: Decentralized Finance (DeFi) attacks. *IEEE S&P*, 2023.
- [5] J. Feist, G. Grieco, A. Groce. Slither: A static analysis framework for smart contracts. *WETSEB*, 2019.
- [6] T. Durieux, J. F. Ferreira, R. Abreu, P. Cruz. Empirical review of automated analysis tools on 47,587 Ethereum smart contracts. *ICSE*, 2020.
- [7] Y. Sun et al. GPTScan: Detecting logic vulnerabilities in smart contracts by combining GPT with program analysis. *ICSE*, 2024.
- [8] Y. Zhuang et al. Smart contract vulnerability detection using graph neural networks. *IJCAI*, 2020.
- [9] MagiChain Security. Price-oracle manipulation: what a pre-deployment check can catch, and what it can't. Research note, October 2026. Incident sources are listed there.
- [10] DeFiHackLabs: reproductions of DeFi incidents with Foundry. <https://github.com/SunWeb3Sec/DeFiHackLabs>