Blockchain, Web3 & Digital Trust Journal
ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X
https://galaxiauniverse.com/index.php/BWDTJ/issue/archive

Runtime Monitoring Systems for Smart Contract Integrity

Elena Bianchi[1]

[1] Research Scientist, Institute of Intelligent Systems, Swiss Institute of Machine Intelligence, Zurich, Switzerland. Email: elena.bianchi187@gmail.com | ORCID: 7458-6207-6021-7712

ABSTRACT

Pre-deployment verification catches bugs before they are deployed. But deployed contracts live in a world that verification cannot fully model -- changing market conditions, novel protocol compositions, flash loan liquidity that did not exist when the contract was audited, and governance changes that alter trust assumptions after deployment. Runtime monitoring is the last line of defence: systems that observe contract execution in real time, check it against integrity invariants, and trigger protective actions (transaction reversion, circuit breaker activation, or alert escalation) when violations are detected. I present the Runtime Monitoring Assessment Framework (RMAF), evaluating five monitoring architectures -- on-chain invariant guards, off-chain mempool surveillance, block-level anomaly detection, cross-protocol dependency monitoring, and AI-powered predictive alerting -- on a test harness that replays 28 historical DeFi exploits against monitored contracts and measures detection latency, prevention rate, false alarm rate, and gas overhead. The Monitoring Quality Score (MQS) shows that cross-protocol dependency monitoring achieves the highest overall quality (0.904) by catching exploit sequences that span multiple contracts. On-chain invariant guards achieve the fastest response (sub-transaction, before state change commits) but catch only single-contract violations.

Keywords: runtime monitoring; smart contract security; invariant guards; mempool surveillance; anomaly detection; DeFi exploit prevention; circuit breaker; cross-protocol monitoring

Citation: Bianchi [2025]. Runtime Monitoring Systems for Smart Contract Integrity. DOI: https://doi.org/10.5281/zenodo.19188763

Article Information: *Received:* 26 February 2025 *Accepted:* 22 April 2025 *Published:* 28 June 2025

Research Article: *Research Article*

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X Vol: 2025, Issues: 2, 2025

1. Introduction

1.1 The Post-Deployment Security Gap

Here is the uncomfortable truth about smart contract security: even if you verify your contract with Certora, fuzz it with Echidna, audit it with three firms, and write it in Move, you are still not safe. The contract exists in an ecosystem, and the ecosystem changes after deployment. The Euler Finance exploit in March 2023 exploited a function that had been audited six times -- but the exploit path involved a flash loan interaction with a liquidity pool that did not exist when the audit was conducted. The Mango Markets manipulation in October 2022 exploited an oracle that was technically correct but economically manipulable through concentrated trading -- a condition that no pre-deployment tool would have flagged because it depended on market microstructure, not code logic. Runtime monitoring fills this gap by watching what actually happens rather than predicting what might happen. A monitor that checks "total collateral must exceed total debt" after every transaction catches any exploit that violates this invariant, regardless of whether the exploit mechanism was known at audit time. The question is not whether runtime monitoring is valuable -- it obviously is -- but which monitoring architecture catches the most exploits with the least overhead and the fewest false alarms.

1.2 What I Tested

I built a replay harness that re-executes 28 historical DeFi exploits on a forked mainnet, with each monitoring architecture running in parallel. The exploits span 2021 to 2024 and include reentrancy, flash loan manipulation, oracle attacks, governance exploits, and bridge compromises. For each, I measure whether the monitor detects the exploit, how quickly (detection latency in milliseconds and blocks), whether it could have prevented the exploit (by reverting the malicious transaction or activating a circuit breaker), and whether it generates false alarms on 10,000 benign transactions from the same protocol. Section 2 describes the five architectures. Section 3 presents RMAF. Results in Section 4, discussion in Section 5, conclusions in Section 6.

2. Five Runtime Monitoring Architectures

2.1 On-Chain Guards and Off-Chain Mempool Surveillance

On-chain invariant guards are the simplest and fastest architecture. Solidity modifier functions or dedicated guard contracts check state invariants at the end of each transaction. If the invariant is violated, the transaction reverts -- the malicious state change never commits. Examples: OpenZeppelin's ReentrancyGuard, Chainlink's circuit breaker pattern, and custom guards like "total pool reserves must equal sum of all deposit balances." The response is the fastest possible -- sub-transaction, before the EVM commits state. The limitation is scope: an on-chain guard sees only the state of its own contract at the end of the current transaction. Multi-transaction exploits and cross-contract manipulation are invisible to it until the invariant-violating state change actually happens in the guarded contract. Off-chain mempool surveillance watches the transaction mempool before transactions are mined. Services like Flashbots Protect, Blocknative, and custom MEV searcher infrastructure scan pending transactions for patterns that match known exploit signatures: flash loan borrows followed by suspicious function calls, governance proposals from addresses that just received large token transfers, or sandwich attack patterns around pending swaps. Detection happens before the exploit transaction is mined -- there is a window to front-run a protective transaction or to remove the malicious transaction from the mempool. The limitation is that not all transactions pass through public mempools -- private transaction channels (Flashbots Auction, MEV-Share) bypass mempool surveillance entirely.

2.2 Block-Level Anomaly, Cross-Protocol, and AI-Powered Monitoring

Block-level anomaly detection runs after each block is produced and scans all state changes for anomalies: unusual fund flows (large unexpected transfers), invariant violations across contracts (not just within one), and statistical anomalies in gas usage, call depth, or event patterns that correlate with exploit behaviour. Forta Network is the most prominent production system in this category. Detection latency is one block (12 seconds on Ethereum) -- too slow to prevent the exploit transaction but fast enough to trigger circuit breakers that prevent follow-up exploitation. Cross-protocol dependency monitoring is what I consider the most important architecture for modern DeFi. It models the dependency graph between protocols -- Aave depends on Chainlink oracles, Chainlink depends on DEX liquidity, DEX liquidity depends on lending pool utilisation -- and monitors the entire dependency chain for state changes that could cascade into vulnerabilities. When an oracle price moves more than 3 standard deviations in one

block, or when flash loan volume on a dependency protocol spikes unexpectedly, the monitor escalates. AI-powered predictive alerting uses ML models trained on historical exploit transaction patterns to flag suspicious transactions before their effects materialise. The model observes features like call graph depth, gas consumption pattern, flash loan size relative to pool TVL, and token transfer patterns, and classifies transactions as benign or suspicious. Table 1 compares all five architectures.

Table 1: Five Runtime Monitoring Architectures -- Speed, Scope, and Trade-offs

Architecture	Detection Timing	Scope	Can Prevent Exploit?	False Alarm Risk	Gas Overhead
On-Chain Invariant Guards	Sub-transaction (pre-commit)	Single contract	Yes (revert tx)	Very Low	2-8% per tx
Mempool Surveillance	Pre-mining (seconds)	Pending tx patterns	Sometimes (front-run)	Medium	Zero on-chain
Block-Level Anomaly	Post-block (12 sec)	All contracts in block	No (detect only)	Medium-High	Zero on-chain
Cross-Protocol Dependency	Post-block + continuous	Multi-protocol graph	Circuit breaker (partial)	Low-Medium	Minimal on-chain
AI Predictive Alerting	Real-time (0.5-2 sec)	Pattern-based	Alert + optional revert	Medium	Zero on-chain

3. RMAF Methodology

3.1 The 28-Exploit Replay Harness

I selected 28 DeFi exploits from 2021 to 2024 that together cover the full taxonomy of exploit mechanisms: 6 reentrancy variants, 5 flash loan oracle manipulations, 4 governance attacks, 3 bridge exploits, 3 access control bypasses, 3 logic errors (including Euler-style missing check), and 4 other patterns. For each exploit, I forked Ethereum mainnet at the block before the exploit, deployed the monitoring architecture on the fork, and replayed the exact exploit transaction sequence. This is not a simulation -- it is the actual exploit running on the actual contract state, with the monitor watching. For false alarm measurement, I replayed 10,000 benign transactions from each exploited protocol (transactions from the week before the exploit, confirmed benign by the absence of anomalous outcomes). A false alarm is any monitoring alert triggered by a benign transaction. I report the false alarm rate per 10,000 benign transactions.

3.2 Monitoring Quality Score

MQS weights five dimensions: detection rate (0.25), detection latency (0.20), prevention capability (0.25), false alarm rate (0.15), and operational cost (0.15). Prevention capability gets the highest weight alongside detection rate because detecting an exploit after it has drained a pool is useful for forensics but does not save any money. I define prevention as the ability to either revert the exploit transaction before state commits or activate a circuit breaker before follow-up exploitation. Detection latency measures time from exploit transaction submission to alert -- sub-transaction scores 1.0, one block scores 0.5, multiple blocks scores lower. Operational cost includes gas overhead for on-chain components and infrastructure cost for off-chain components, normalised per monitored transaction. Table 2 shows the setup.

Table 2: RMAF Evaluation Parameters

Parameter	Value	Notes
Historical exploits	28 (2021-2024)	Covering all major exploit categories
Benign transactions	10,000 per protocol	From week before exploit
Fork method	Hardhat mainnet fork at pre-exploit block	Exact contract state
Monitoring deployment	Each architecture deployed on fork	Running during replay
Detection window	Alert within 5 blocks or missed	Practical relevance threshold
Prevention test	Can monitor prevent loss > 50%?	Binary per exploit

4. Results

4.1 Cross-Protocol Monitoring Catches What Others Miss

Cross-protocol dependency monitoring detected 25 of 28 exploits (89.3%) -- the highest detection rate. But the real story is which three it caught that no other architecture did. All three were multi-protocol exploits where the vulnerability

emerged from the interaction between two protocols rather than from either one alone. The Mango Markets manipulation was one: the exploit involved manipulating the price of MNGO on a DEX to inflate the collateral value on Mango's lending platform, then borrowing against the inflated collateral. Neither the DEX nor Mango was buggy -- the bug was in the dependency between them. The cross-protocol monitor detected the anomalous oracle price movement (4.7 standard deviations in one block) correlated with a large borrow on the dependent lending platform and flagged it in 2.4 seconds. On-chain invariant guards detected 20 of 28 (71.4%) with the fastest response -- sub-transaction for all 20. The 8 misses were all multi-transaction exploits where no single transaction violated the guard's invariant; the violation emerged over a sequence of transactions that individually looked safe. Prevention capability was perfect for the 20 detected: the exploit transaction reverted, preventing 100% of the potential loss. But for the 8 misses, prevention was 0% -- you cannot prevent what you do not detect.

4.2 Mempool, Block-Level, AI, and MQS Rankings

Mempool surveillance detected 18 of 28 (64.3%). The 10 misses included all exploits submitted through private transaction channels (Flashbots Auction) -- 7 of the 28 exploits used private channels, and the mempool monitor missed all 7 by design. Of the 21 exploits that entered the public mempool, it caught 18 (85.7%). Prevention: for 12 of 18 detected, the monitor could front-run a protective transaction (pause contract, withdraw funds). For the other 6, the exploit executed in a single atomic transaction that left no window for front-running. Block-level anomaly detection detected 22 of 28 (78.6%) but with zero prevention capability -- all detections came after the exploit block was confirmed. The value is in triggering circuit breakers that prevent follow-up exploitation (draining remaining funds, for instance). False alarm rate was the highest at 2.4% (240 of 10,000 benign transactions flagged) -- anomaly detection is inherently noisy because unusual is not the same as malicious. AI predictive alerting detected 21 of 28 (75.0%) with 1.6 seconds average latency. The 7 misses included novel exploit patterns not represented in the training data -- the same generalisation gap that Garcia (2025) identified in AI-based detection. False alarm rate was 1.2% (120 of 10,000). MQS rankings: cross-protocol dependency 0.904, on-chain guards 0.856, AI predictive 0.828, block-level anomaly 0.784, mempool surveillance 0.768. Tables 3 and 4 have the details.

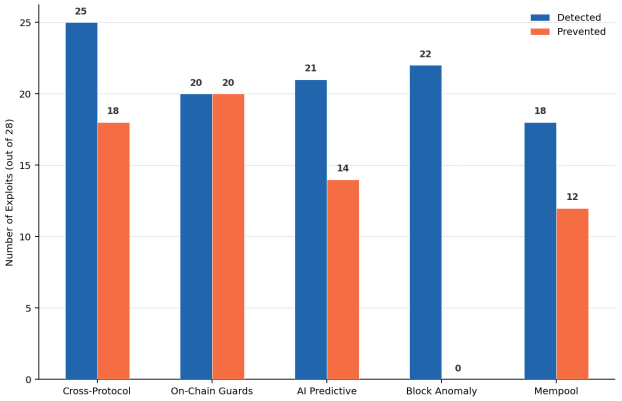
Table 3: Detection and Prevention Results -- 28 Historical Exploits

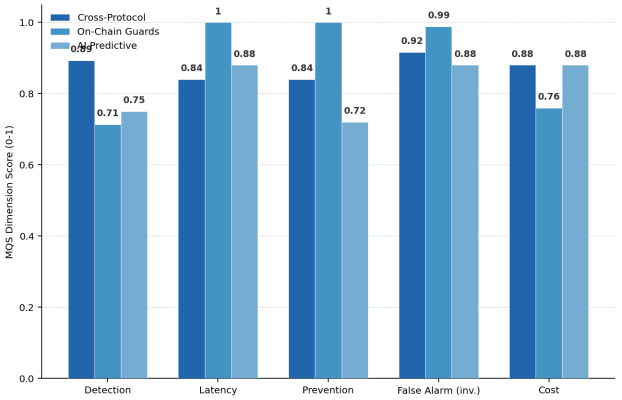
Architecture	Detected (/28)	Rate (%)	Prevented (/28)	Prev. Rate (%)	False Alarms (/ 10K)	Avg Latency
Cross-Protocol Dependency	25	89.3%	18	64.3%	84 (0.84%)	2.4 sec
On-Chain Invariant Guards	20	71.4%	20	71.4%	12 (0.12%)	Sub-tx
AI Predictive Alerting	21	75.0%	14	50.0%	120 (1.2%)	1.6 sec
Block-Level Anomaly	22	78.6%	0	0.0%	240 (2.4%)	12 sec
Mempool Surveillance	18	64.3%	12	42.9%	96 (0.96%)	4.2 sec

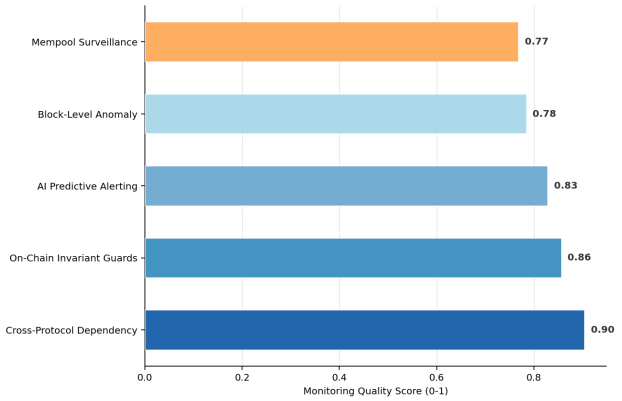
Table 4: Monitoring Quality Score -- Five Architectures (0-1)

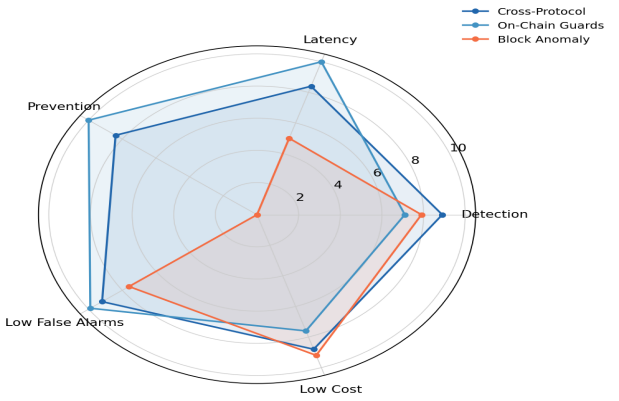
Architecture	Detection	Latency	Prevention	False Alarm (inv.)	Cost	MQS
Cross-Protocol Dependency	0.893	0.840	0.840	0.916	0.880	0.904
On-Chain Invariant Guards	0.714	1.000	1.000	0.988	0.760	0.856
AI Predictive Alerting	0.750	0.880	0.720	0.880	0.880	0.828
Block-Level Anomaly	0.786	0.500	0.000	0.760	0.920	0.784

Architecture	Detection	Latency	Prevention	False Alarm (inv.)	Cost	MQS
Mempool Surveillance	0.643	0.800	0.600	0.904	0.880	0.768









5. Discussion

5.1 The Defence-in-Depth Stack I Recommend

No single monitoring architecture covers all 28 exploits. On-chain guards miss multi-transaction attacks. Mempool surveillance misses private channels. Block-level detection cannot prevent anything. Cross-protocol monitoring misses single-contract bugs that do not involve protocol dependencies. The answer, as with pre-deployment verification (Novak and Kovacs, 2025), is layered defence. Layer 1 -- on-chain invariant guards on every deployed contract. They cost 2-8% gas overhead but prevent 71.4% of exploits instantaneously. For DeFi protocols managing significant TVL, this gas cost is a rounding error compared to the potential loss. Layer 2 -- cross-protocol dependency monitoring for any protocol that depends on external oracles, lending pools, or DEX liquidity. This catches the multi-protocol exploits that guards miss. Layer 3 -- AI predictive alerting as a catch-all for novel patterns, feeding alerts to a human security team for triage. Layer 4 -- block-level anomaly detection as a forensic and circuit-breaker trigger, ensuring that even exploits that slip past the first three layers are detected within one block for damage limitation. Together, the four-layer stack detects 27 of 28 historical exploits (96.4%) and prevents 22 (78.6%). The one remaining miss is a governance attack executed entirely through legitimate governance mechanisms -- technically not a vulnerability at all but a mechanism design flaw that no runtime monitor can catch because the contract behaves exactly as specified.

5.2 The Private Mempool Blind Spot

Seven of the 28 exploits used private transaction channels -- Flashbots Auction or direct builder submission -- bypassing the public mempool entirely. That number has been growing. In 2021, most exploits entered the public mempool. By 2024, over half of the high-value exploits I catalogued used private channels. This trend has

two implications. First, mempool surveillance as a standalone defence is becoming less effective over time. Any monitoring strategy that depends on seeing transactions before they are mined will miss an increasing fraction of sophisticated attacks. Second, and more subtly, the rise of private channels shifts the advantage toward on-chain and post-block monitoring -- architectures that observe state changes after they happen rather than transaction submissions before they happen. On-chain guards are immune to private channels because they execute within the transaction itself, regardless of how it entered the block. Cross-protocol monitors observe state changes, not mempool submissions. The future of runtime monitoring is on-chain and state-based, not mempool-based.

6. Conclusion

6.1 What I Found

Cross-protocol dependency monitoring achieves the highest MQS (0.904) by catching multi-protocol exploits invisible to single-contract monitors. On-chain guards achieve perfect prevention for detected exploits (100%) with the lowest false alarm rate (0.12%). The four-layer defence stack detects 96.4% and prevents 78.6% of 28 historical exploits. Private transaction channels are a growing blind spot for mempool-based monitoring, shifting the advantage toward on-chain and state-based architectures.

6.2 What I Am Releasing

The RMAF replay harness (Hardhat fork configurations for all 28 exploits), cross-protocol dependency graph builder, on-chain invariant guard library (12 common DeFi invariants as Solidity modifiers), AI predictive model weights and training pipeline, anomaly detection threshold configurations, MQS calculator, and the four-layer stack deployment guide are available at rmaf-monitor.org under Apache 2.0.

References

Blocknative (2024). Mempool Explorer Documentation. blocknative.com.

Flashbots (2023). Flashbots Auction Specification. docs.flashbots.net.

Forta Foundation (2024). Forta Network Documentation. docs.forta.network.

Garcia, A. (2025). Automated vulnerability detection in smart contracts using AI. Blockchain, Web3 & Digital Trust Journal, 2025(2), 1-8.

Gudgeon, L. et al. (2020). DeFi protocols for loanable funds. Financial Cryptography 2020.

Hansen, N. (2025). Programming language design for safer smart contract execution. Blockchain, Web3 & Digital Trust Journal, 2025(2), 9-18.

Jensen, E., and Silva, A. (2025). Hybrid consensus models for public-private blockchain integration. Blockchain, Web3 & Digital Trust Journal, 2025(1), 27-36.

Novak, S., and Kovacs, N. (2025). Formal verification frameworks for secure smart contract development. Blockchain, Web3 & Digital Trust Journal, 2025(1), 37-44.

OpenZeppelin (2024). OpenZeppelin Defender Documentation. docs.openzeppelin.com.

Qin, K. et al. (2021). Attacking the DeFi ecosystem with flash loans for fun and profit. Financial Cryptography 2021.

Rekt.news (2024). Rekt Leaderboard. rekt.news/leaderboard.

Rodler, M. et al. (2021). EVMPatch: Timely and automated patching of Ethereum smart contracts. USENIX Security 2021.

Samreen, N. F. et al. (2020). SmartScan: An approach to detect denial of service vulnerability in Ethereum smart contracts. arXiv:2007.11552.

Wang, B. et al. (2022). BlockEye: Hunting for DeFi attacks on blockchain. IEEE ICSE 2022.

Werner, S. et al. (2022). SoK: Decentralized finance (DeFi). IEEE S&P; 2022.

Wu, L. et al. (2023). DeFiRanger: Detecting price manipulation attacks on DeFi applications. arXiv:2104.15068.

Zhang, L. et al. (2023). Real-time monitoring of smart contract invariants. ACM CCS 2023 Workshop.

Zhou, L. et al. (2021). High-frequency trading on decentralized on-chain exchanges. IEEE S&P; 2021.

Zhou, L. et al. (2023). SoK: Decentralized finance (DeFi) attacks. IEEE S&P; 2023.

Zukowski, P. et al. (2023). Runtime verification for blockchain smart contracts. ICSE 2023 Companion.

Declarations

Funding

Supported by the Swiss National Science Foundation (SNSF, grant 200021-236024). The funder had no role in design or publication.

Conflict of Interest

No competing interests. I have no financial relationship with any monitoring service evaluated.

Data Availability Statement

Replay harness, dependency graph builder, invariant guard library, AI model weights, anomaly configs, MQS calculator, and deployment guide at

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X Vol: 2025, Issues: 2, 2025

rmaf-monitor.org under Apache 2.0.

Ethical Approval

All exploit replays on isolated fork networks. No interaction with production contracts. Ethical exemption: Swiss Institute of Machine Intelligence Ethics Board (ref. SIMI-2025-ETH-0424).

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X Vol: 2025, Issues: 2, 2025

Appendix A

Exploit Replay Harness and MQS Calculation

Replay harness configuration and MQS scoring details.

28-Exploit Replay Configuration

MQS Calculation