Olympix

EXPLOIT RESEARCH · 2025 REVIEW

2025 EVM Smart Contract Exploit Analysis

Fifty in-scope exploits from 2025, and whether the vulnerable logic was already in the code before deployment.

49 / 50



In-scope 2025 EVM exploits Olympix would have prevented before deployment in a retrospective backtest, roughly $247M in losses.

Contents

01 Executive summary 03
What the 2025 dataset shows.

02 2025 EVM smart contract exploits 04
Scope, the full list of 50, and preventability.

03 Distribution of vulnerability types 08
Frequency versus dollar impact.

04 Top 3 exploits by economic impact 10
Balancer, GMX and Cork Protocol.

05 Analysis and risk implications 14
Volume versus losses, and twelve findings.

06 What 2025's data reveals 18
Four conclusions on security maturity.

07 2026 recommendations for enterprise teams 19
Layered security, shifting left, and reproducible findings.

About Olympix 23

01

Executive summary

A structured review of every in-scope 2025 EVM exploit

This report presents a structured review of EVM smart contract exploits that occurred throughout 2025, with the objective of assessing preventability, identifying systemic risk patterns, and evaluating what the data reveals about the current maturity of crypto security practices.

The analysis focuses on exploits arising from vulnerabilities in Solidity-based smart contracts deployed within the EVM ecosystem. These incidents represent failures in logic, accounting, access control, or invariant enforcement, rather than offchain compromise, key theft, or social engineering.

A central theme of this report is preventability.

In this context, an exploit is considered preventable when the vulnerable logic existed in contract code prior to deployment and could have been identified through deterministic security testing during development.

Across the in-scope dataset, the findings are unambiguous.

49 of 50

In-scope EVM exploits Olympix would have prevented before deployment.

~98%

Of in-scope incidents.

~$247M

In preventable losses.

Of the 50 EVM smart contract exploits reviewed, Olympix would have prevented 49, representing approximately 98 percent of incidents and roughly $247 million in preventable losses. Only a single exploit fell outside the scope of vulnerabilities detectable prior to deployment.

These results underscore a broader conclusion. The majority of losses in 2025 were not the result of novel attack techniques, but of vulnerabilities introduced earlier in development and left undiscovered until exploitation.

This report documents those failures, examines their implications, and outlines how enterprise teams can evolve their security posture heading into 2026.

02

2025 EVM smart contract exploits

In-scope exploits

This review focuses exclusively on exploits arising from vulnerabilities in EVM smart contracts written in Solidity.

Only incidents in which loss resulted from flaws in onchain contract logic were included in the analysis. The objective of this scoping was to isolate failures attributable to smart contract correctness, rather than operational, human, or infrastructure compromise.

In scope

An exploit was considered in scope if:

- ✓ The affected protocol operated within the EVM ecosystem
- ✓ The vulnerable component was a Solidity-based smart contract
- ✓ The exploit resulted from logic, accounting, access control, or invariant failures
- ✓ The vulnerable code existed prior to deployment
- ✓ The exploit was executed through valid onchain transactions

Out of scope

Losses that resulted from factors unrelated to smart contract correctness, including:

- × Private key compromise or wallet leakage
- × Social engineering or phishing attacks
- × Infrastructure or backend system breaches
- × Exploits occurring on non-EVM chains
- × Exit scams or incidents where contract code was unavailable for review

This scoping ensures the analysis reflects only those failures that could reasonably be detected and prevented through deterministic smart contract security testing during development.

The complete list of the 2025 exploits analyzed follows, in date order.

02

The 50 exploits reviewed

Every in-scope 2025 EVM incident in this analysis, in date order (1 to 25).

Project	Date	Technique	Lost	Prevented?
Sorra Finance	01/04/2025	Logic Vulnerability	**$41K**	Yes
Mosca	01/08/2025	Logic Vulnerability	**$19.5K**	Yes
FortuneWheel	01/10/2025	Access Control	**$21K**	Yes
UniLend Finance	01/13/2025	Logic Vulnerability	**$200K**	Yes
Idols NFT	01/15/2025	Reward Calculation Error	**$340K**	Yes
AST Token	01/21/2025	Logic Vulnerability	**$65K**	Yes
ODOS Protocol	01/23/2025	Insufficient Input Validation	**$50K**	Yes
BankX	02/08/2025	Reentrancy	**$43K**	Yes
Hegic Options	02/23/2025	Logic Vulnerability	**$80K**	Yes
Venus	02/27/2025	Price Manipulation	**$716K**	No
Zoth	03/01/2025	Logic Vulnerability	**$285K**	Yes
1inch	03/05/2025	Logic Vulnerability	**$5M**	Yes
H2O Token	03/14/2025	Logic Vulnerability	**$22K**	Yes
WebKey	03/14/2025	Arbitrage	**$737K**	Yes
BBX	03/20/2025	Price Manipulation	**$12K**	Yes
Alkimiya	03/28/2025	Logic Vulnerability	**$96K**	Yes
SIR	03/30/2025	Logic Vulnerability	**$355K**	Yes
KiloEx	04/14/2025	Unauthorized Access	**$7.4M**	Yes
Numa	04/18/2025	Price Manipulation	**$527K**	Yes
Zora	04/24/2025	Access Control	**$128K**	Yes
Impermax	04/26/2025	Price Manipulation	**$300K**	Yes
MBU	05/11/2025	Precision Loss	**$2.2M**	Yes
Nitron	05/16/2025	Price Manipulation	**$951K**	Yes
LNDFi	05/16/2025	Unauthorized Access	**$1.3M**	Yes
Dexodus	05/26/2025	Signature Replay	**$291K**	Yes

The 50 exploits reviewed, continued (26 to 50).

Project	Date	Technique	Lost	Prevented?
Cork Protocol	05/28/2025	Missing Authorization Checks	**$12M**	Yes
Meta Pool	06/17/2025	Missing Override	**$137K**	Yes
ResupplyFi	06/26/2025	Price Manipulation	**$9.6M**	Yes
Future Protocol	07/02/2025	Price Manipulation	**$4.6M**	Yes
Rant	07/05/2025	Price Manipulation	**$204K**	Yes
GMX	07/09/2025	Price Manipulation	**$42M**	Yes
WETC Token	07/17/2025	Logic Vulnerability	**$100K**	Yes
SuperRare	07/28/2025	Access Control	**$680K**	Yes
Numa	08/10/2025	Logic Vulnerability	**$313K**	Yes
Betterbank	08/27/2025	Logic Vulnerability	**$5M**	Yes
Bunni	09/02/2025	Rounding Error	**$8.4M**	Yes
Kame Aggregator	09/12/2025	Missing Validation Check	**$1.3M**	Yes
New Gold Protocol	09/18/2025	Price Manipulation	**$2M**	Yes
Griffin AI	09/25/2025	Misconfigured Deployment	**$3M**	Yes
dTrinity	09/28/2025	Access Control	**$56K**	Yes
Abracadabra	10/04/2025	Logic Vulnerability	**$1.7M**	Yes
Typus Finance	10/15/2025	Missing Authorization Checks	**$3.4M**	Yes
Sharwa Finance	10/20/2025	Missing Slippage Protection	**$147K**	Yes
Balancer	11/03/2025	Precision Loss	**$121.1M**	Yes
Moonwell	11/04/2025	Faulty Price Oracle	**$1M**	Yes
DRLVaultV3	11/10/2025	Slippage Exploit	**$98K**	Yes
Port3	11/23/2025	Logic Vulnerability	**$166K**	Yes
Yearn Finance	11/30/2025	Logic Vulnerability	**$9M**	Yes
DMi Token	12/08/2025	Logic Vulnerability	**$124K**	Yes
Prism Protocol	12/11/2025	Access Control & Price Manipulation	**$62K**	Yes

02

Preventability summary

Across the 50 in-scope EVM exploits reviewed



Preventable with Olympix (49) Not preventable (1)

49

Exploits were preventable using Olympix prior to deployment.

98%

Of incidents fell within detectable vulnerability classes.

~$247M

In losses were preventable.

These findings reflect a consistent pattern across the year. Vulnerable logic was present in deployed contracts well before exploitation occurred, often long before audits or monitoring tools could intervene.

The single incident outside that scope was Venus (February 27, 2025, price manipulation, $716K).

03

Distribution of vulnerability types

Most commonly exploited, by count

A small set of vulnerability categories accounted for the majority of exploit activity.

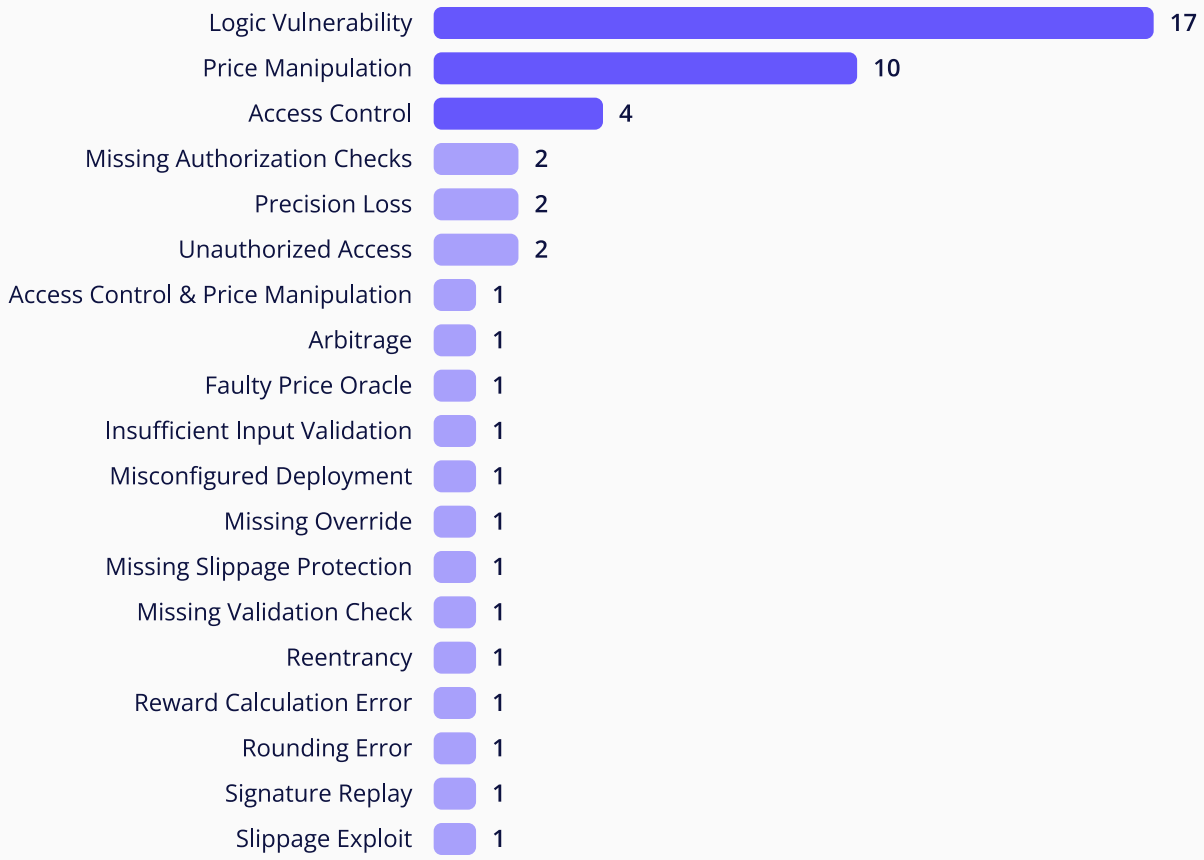
Logic vulnerabilities were the most frequently exploited category in 2025. These failures typically involved:

- incorrect assumptions about state transitions
- unsafe ordering of operations
- multi-step execution paths that violated intended behavior

Attackers did not bypass permissions or exploit infrastructure weaknesses. They used valid function calls.

The protocol behaved exactly as written.

Vulnerability categories by number of exploits (2025)



03

Distribution of vulnerability types

Highest total dollar losses

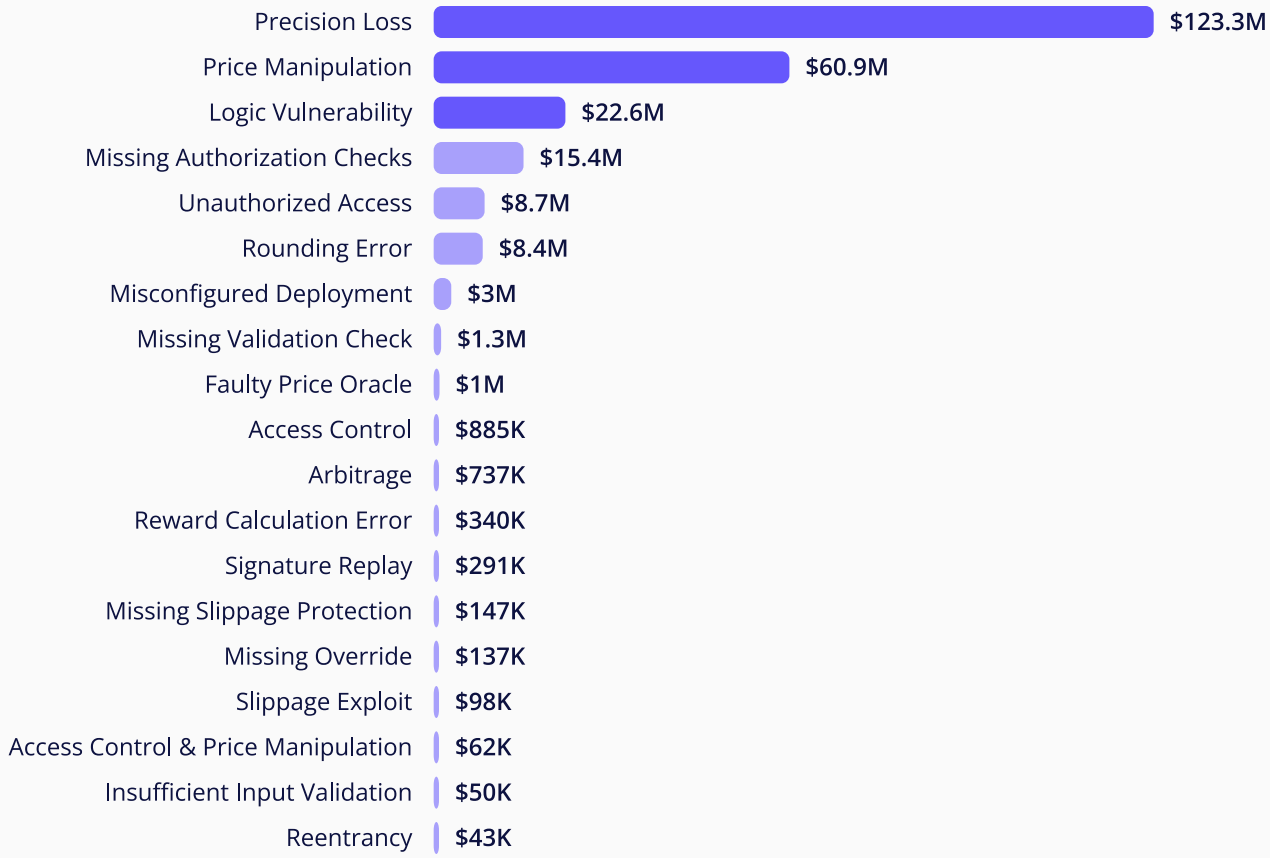
Accounting and precision errors were responsible for the highest total dollar losses.

Although fewer in number, these vulnerabilities had severe consequences due to their presence in high liquidity systems. Small numerical errors scaled catastrophically when applied repeatedly.

Frequency ≠ impact

This imbalance between frequency and impact is one of the most important lessons of 2025.

Vulnerability categories by total dollar impact (2025)

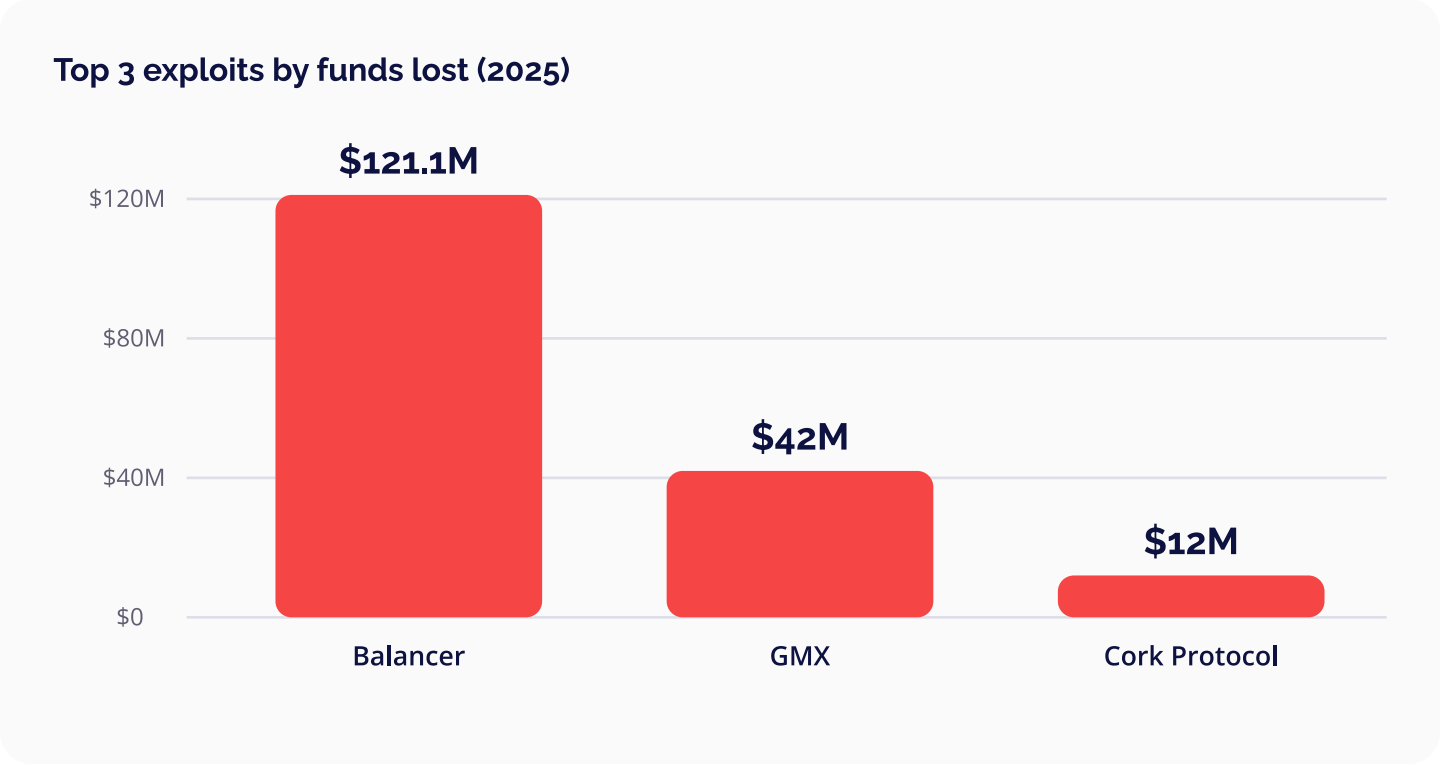


04

Top 3 exploits by economic impact

A small subset, a disproportionate share of losses

A small subset of incidents accounted for a disproportionate share of annual losses. Each of these exploits stemmed from deterministic logic failures that existed prior to audit and deployment.



Balancer

Balancer V2

$121.1M

Nov 3, 2025
Precision loss

GMX

GMX V1

$42M

Jul 9, 2025
Cross-contract reentrancy

CORK

Cork Protocol

$12M

May 28, 2025
Missing access control

The post-mortems for each follow.

04 · Post-mortem

Balancer V2

Balancer

Loss $121.1M | November 3, 2025 | Precision loss | Preventable with Olympix

Balancer V2 suffered one of the largest exploits of 2025 after attackers abused a precision and rounding flaw in Composable Stable Pools. Through carefully calibrated micro-swaps, attackers were able to erode pool invariants and extract value across multiple networks.

The exploit was not caused by oracle failure, governance compromise, or privileged access. It originated entirely from deterministic accounting logic.

Root cause

Asymmetric rounding in rate-augmented scaling logic.

Scaling functions applied consistent downward rounding while embedding dynamic token rates. Under specific pool conditions, this introduced a persistent rounding bias that allowed attackers to underpay during EXACT_OUT swaps. Repeated over many operations, this bias degraded the pool invariant and deflated BPT value.

```
function _upscale(uint256 amount, uint256 scalingFactor) internal pure returns (uint256) {
    // Upscale rounding wouldn't necessarily always go in the same direction: in a swap
    // for example the balance of token in should be rounded up, and that of token out
    // rounded down. This is the only place where we round in the same direction for
    // all amounts, as the impact of this rounding is expected to be minimal (and
    // there's no rounding error unless `_scalingFactor()` is overriden).
    return FixedPoint.mulDown(amount, scalingFactor);
}
```

How Olympix would have prevented this

Olympix's precision analysis would have flagged biased rounding in rate-augmented scaling factors as a High-severity vulnerability.

The system would flag:

- inconsistent rounding direction across scaling operations
- amplification risk when dynamic rates are embedded
- invariant degradation under adversarial pool states

This class of vulnerability is extremely difficult to surface through manual review, but well-suited to automated, deterministic analysis.

04 · Post-mortem

GMX V1

GMX

Loss $42M | July 9, 2025 | Cross-contract reentrancy | Preventable with Olympix

In July 2025, GMX V1 suffered a $42M exploit caused by a design flaw in its position execution flow. A user-controlled receiver address enabled a cross-contract reentrancy attack that bypassed routing safeguards and corrupted price calculations used to value GLP.

Root cause

Unconstrained user-controlled receiver execution.

The decreasePosition flow allowed users to specify an arbitrary receiver address. When this receiver was a contract, execution control was transferred externally before all internal state transitions and invariants were finalized.

This enabled cross-contract reentrancy into sensitive vault functions, bypassing expected routing through GMX's PositionRouter and PositionManager.

```
function executeDecreaseOrder(address _address, uint256 _orderIndex,
    address payable _feeReceiver) external nonReentrant {
    ...
    // transfer released collateral to user
    if (order.collateralToken == weth) {
        _transferOutETH(amountOut, payable(order.account));
    } else {
        IERC20(order.collateralToken).safeTransfer(order.account, amountOut);
    }
    ...
function _transferOutETH(uint256 _amountOut, address payable _receiver) private {
    IWETH(weth).withdraw(_amountOut);
    _receiver.sendValue(_amountOut);
}
```

How Olympix would have prevented this

Olympix's source-level analysis would have flagged the dynamic receiver execution path as exploitable.

The system identifies:

- user-controlled external call sites prior to invariant finalization
- cross-contract reentrancy exposure beyond local guards
- sequencing violations where funds or execution control are transferred prematurely

Through mutation testing and invariant validation, Olympix simulates mid-transaction reentry and validates that price and exposure assumptions cannot be violated under adversarial execution.

04 · Post-mortem

Cork Protocol

Loss $12M | May 28, 2025 | Missing access control | Preventable with Olympix

In May 2025, Cork Protocol suffered a $12M exploit due to critical weaknesses in its Uniswap V4 hook implementation. Missing access control allowed attackers to invoke hook logic directly, mint derivative tokens without depositing collateral, and redeem them for valuable assets.

Root cause

Missing access control and insufficient validation in Uniswap V4 hooks.

Critical hook callbacks, including beforeSwap, lacked restrictions ensuring they could only be invoked by the Uniswap V4 PoolManager. This allowed arbitrary contracts to call hook logic directly with attacker-controlled parameters.

Additional weaknesses in token-type validation and market configuration compounded the exploit.

```
function beforeSwap(
    address sender,
    PoolKey calldata key,
    IPoolManager.SwapParams calldata params,
    bytes calldata hookData
) external override returns (bytes4, BeforeSwapDelta delta, uint24)
```

How Olympix would have prevented this

Olympix's source-level analysis would have flagged the hook callbacks as unsafe due to missing PoolManager-only access enforcement.

The system identifies:

- externally callable hook functions lacking trusted caller validation
- execution paths where hook logic assumes correct invocation context
- missing invariants around asset deposits and derivative issuance

Through mutation testing and invariant validation, Olympix simulates adversarial hook invocations and verifies that minting logic cannot be triggered without actual collateral movement.

05

Analysis and risk implications

Two clear and contrasting trends

Smart contract exploits in 2025 followed two clear and contrasting trends.

Exploit activity was distributed relatively evenly across the year, while total financial losses were heavily concentrated in a small number of high-impact incidents.

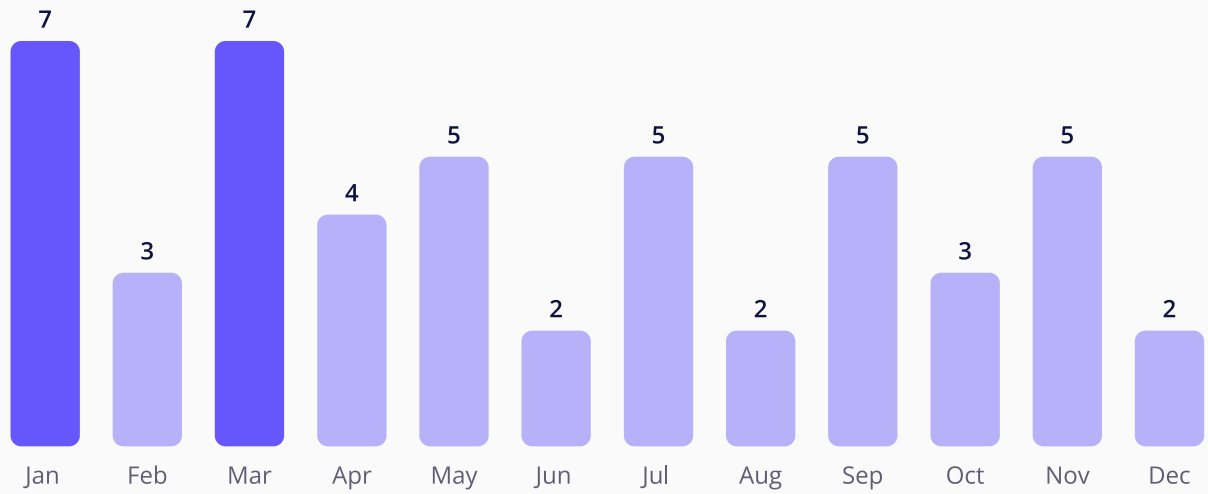
This divergence between frequency and severity defines the 2025 risk landscape.

Biggest months by exploit volume

January and March

January and March tied for the most independent exploit events in 2025, with seven incidents each.

Exploits by month (2025)



Most of January's incidents were logic vulnerabilities. While individual losses were relatively small compared to later incidents, their frequency illustrates how often preventable vulnerabilities continue to ship into production.

These months reflect the most common failure mode of 2025, frequent, low-to-mid impact exploits caused by unsafe assumptions in contract logic.

05

Hacks by volume vs. hacks by losses

Biggest month by funds lost

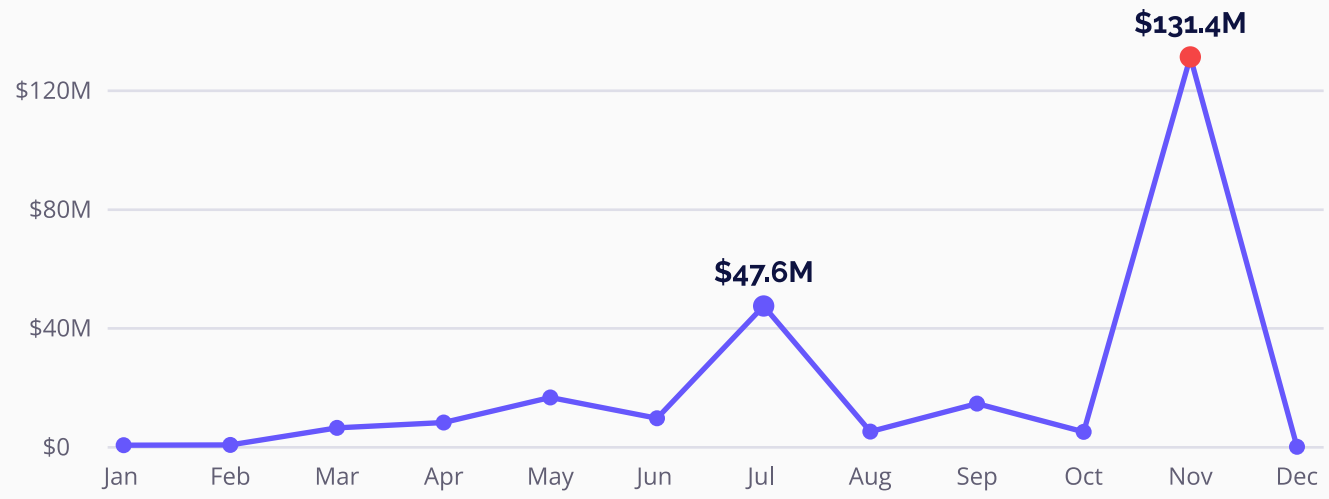
November

November accounted for the largest share of funds lost in 2025, $131.4M across five incidents, fewer than January or March.

The spike was driven by a single exploit, Balancer at $121.1M, which on its own outweighed the combined losses of every other month of the year.

The data shows that exploit severity does not correlate with exploit frequency. One accounting or precision failure in a high-liquidity system can eclipse dozens of smaller incidents.

Total funds lost by month (2025)



The month with the most exploits was not the month with the biggest losses.

Incident volume alone fails to capture systemic risk, as fewer but more severe exploits can dominate outcomes.

05

Risk implications

Twelve findings, and why each one matters

1 Most losses were caused by unmeasured assumptions

The majority of high-impact exploits did not stem from unknown vulnerability classes, but from implicit assumptions about how contracts would be used, composed, or interacted with. When assumptions are not explicitly modeled or tested, risk remains invisible until exploitation occurs.

2 Audits review each scope in isolation

Audits are scoped to the change in front of them, and each scope is reviewed on its own. Many exploited paths sat where an audited component met the rest of the system, or where two separately audited components interacted, a seam no single engagement was asked to examine.

3 High test coverage does not equal adversarial coverage

Many exploited execution paths were technically covered by tests but never meaningfully exercised under adversarial conditions. Coverage metrics alone fail to measure whether critical invariants actually hold under hostile inputs.

4 Security confidence is often assumed, not measured

Many teams rely on qualitative confidence derived from audits or test completion rather than measurable, reproducible guarantees. Without reproducible evidence, residual risk cannot be meaningfully evaluated or compared.

5 Manual review does not scale with deployment velocity

As deployment frequency and composability increase, reliance on human review introduces coverage gaps that grow with scale. Manual processes cannot match the pace of modern smart contract development.

6 Vulnerabilities are introduced long before they are found

Most vulnerabilities originated during design or early implementation, yet were found only after integration or deployment, if they were found at all. Controls that run downstream cannot close a gap that opened upstream.

7 Security failures compound upstream

Vulnerabilities discovered later in the lifecycle are harder to remediate safely. Late discovery increases the likelihood of rushed fixes, partial mitigations, or acceptance of known risk in production systems.

8 Risk is highly concentrated, not evenly distributed

Losses followed a fat-tailed distribution, with a small number of incidents accounting for a disproportionate share of total economic impact. This makes average exploit size or incident count unreliable indicators of true exposure.

9 Exploit frequency and economic severity are decoupled

The month with the highest number of exploits was not the month with the highest total losses. Incident volume alone fails to capture systemic risk, as fewer but more severe exploits can dominate outcomes.

10 Economic impact is decoupled from code complexity

Contract size or architectural complexity did not reliably correlate with exploit severity. Exposure was more strongly driven by value concentration than by lines of code or perceived technical sophistication.

11 Business logic failures dominate high-impact exploits

The most damaging incidents resulted from violations of intended economic behavior rather than syntactic errors. This reinforces that correctness cannot be inferred from compilation success or pattern-based detection alone.

12 Post-deployment monitoring is inherently reactive

Onchain monitoring detects exploitation after vulnerable logic has already been deployed and funds are at risk. While essential for response and visibility, monitoring alone cannot prevent first-order losses.

06

What 2025's data reveals about security maturity

Several conclusions emerge clearly from the 2025 exploit data.

1

Most losses did not stem from unknown vulnerabilities.

They resulted from assumptions that were never explicitly tested, measured, or enforced. In many cases, protocol logic behaved exactly as written, but not as intended. When assumptions remain implicit, they become invisible sources of risk.

2

Audits remain essential, but each one covers only the scope it was given.

Audits are batched to a release or a feature and validated in isolation. Exploits emerge at the boundaries, where an audited scope meets the broader system or where two independently audited scopes interact. Audits stay a necessary layer, but they were never designed to be the only place correctness gets checked.

3

High test coverage does not equate to adversarial coverage.

Many exploited execution paths were technically covered by tests, yet never exercised under adversarial conditions. Coverage metrics measure whether code is executed, not whether it behaves safely when inputs are hostile or states are extreme.

4

Security failures compound as they move downstream.

The later a vulnerability is discovered in the lifecycle, the more costly and risky it becomes to remediate. Issues identified after deployment often require mitigations, compensating controls, or emergency response, rather than clean correction. Prevention upstream consistently delivers the highest risk reduction.

The 2025 data indicates that the industry remains in an early stage of security maturity. Security practices are largely focused on detection and response, rather than prevention and correctness. Assurance is derived from process completion (audits, test coverage, reviews) rather than from deterministic validation of intended behavior.

Until security outcomes become measurable and reproducible, risk will continue to scale faster than protection.

07

2026 recommendations for enterprise teams

Beyond audits: a multilayered security approach

The 2025 exploit data makes one conclusion unavoidable: no single security control is sufficient to protect complex smart contract systems operating at scale.

Proactive tooling, audits, monitoring, and incident response each play an important role, but they address different phases of risk. Effective security posture emerges not from relying on one mechanism, but from combining complementary controls across the full lifecycle.

1 Anchor security in development.

The highest leverage security decisions occur during development, before code is deployed or reviewed. Deterministic security tools applied at this stage provide the strongest form of assurance because they evaluate correctness while assumptions are still malleable.

2 Use audits as independent validation, not primary discovery.

Audits remain an essential component of any serious security program, particularly for independent review and external assurance. Enterprise teams should treat audits as a confirmation layer rather than a discovery mechanism, ensuring that the most critical logic has already been validated prior to review. This approach improves audit outcomes, reduces friction, and shifts audit conversations from reactive remediation to risk confirmation.

3 Rely on monitoring for detection, not prevention.

Onchain monitoring systems provide visibility once contracts are live. They are indispensable for detecting abnormal behavior, alerting teams, and reducing blast radius during active incidents. However, monitoring is inherently reactive. It observes consequences, not causes. Enterprise teams should view monitoring as a response acceleration mechanism, not a substitute for upstream correctness.

4 Integrate risk and intelligence for contextual awareness.

These systems enhance situational awareness, support investigations, and assist with compliance and reporting obligations. They complement technical security controls but do not replace them. Their value increases significantly when paired with strong preventive controls upstream.

5 Prepare explicitly for incident response and recovery.

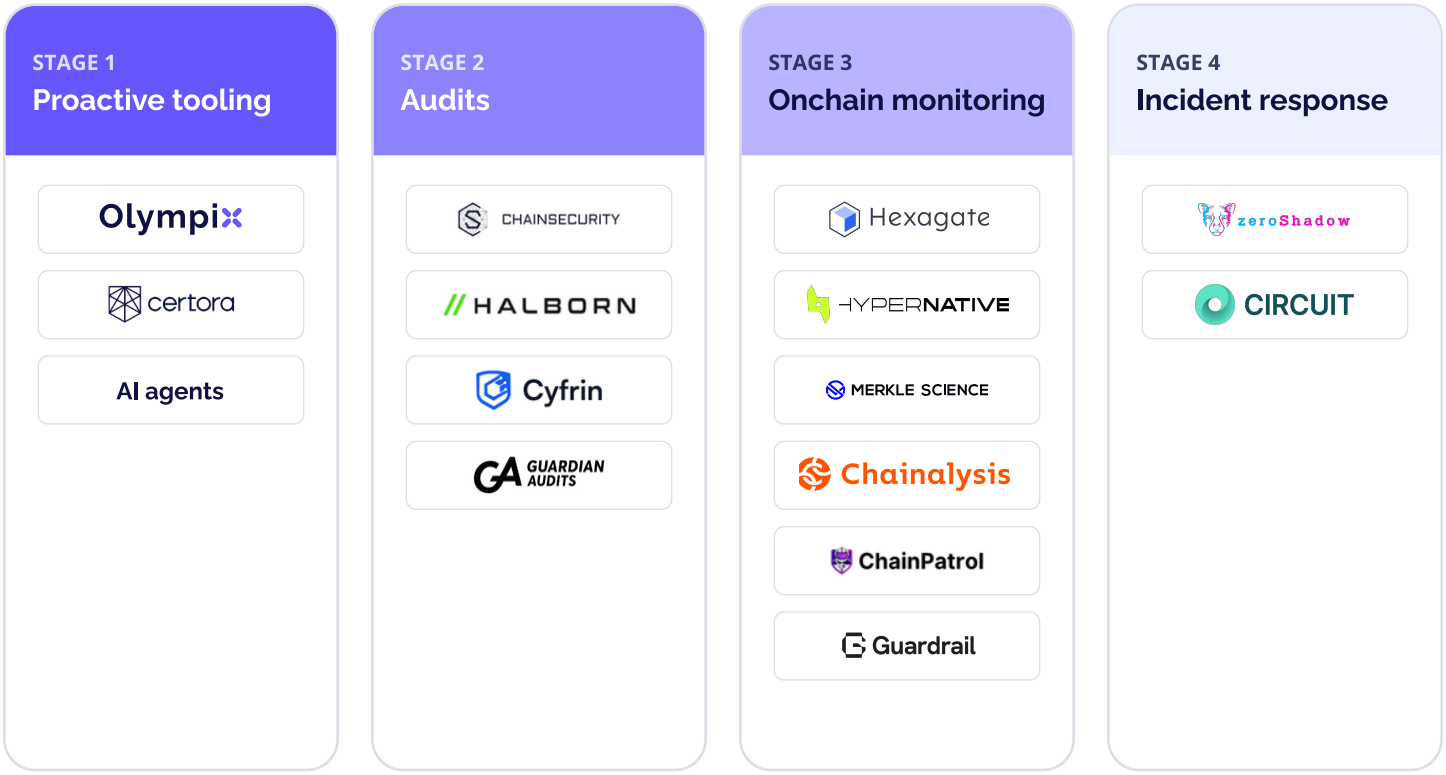
Even mature security programs must assume that incidents can occur. Defined response playbooks, trusted recovery partners, and clear internal ownership are essential for minimizing damage when failures happen. However, response capabilities should be treated as a last line of defense, not a risk strategy. Recovery does not negate loss, reputational damage, or operational disruption. The objective remains reducing the likelihood of incidents reaching this stage at all.

07

Recommended providers by stage

The security lifecycle, from development to recovery

Each stage of the lifecycle addresses a different phase of risk. These are recommended providers for each stage, with proactive tooling at the point where code is written and incident response as the last line of defense.



Areta

Areta evaluates proactive tooling and audit firms to provide whitelists of providers in key ecosystems such as Uniswap and Polygon.

07

Operationalizing security earlier in the SDLC

As markets mature, security spend shifts left

As software systems mature and economic risk increases, security consistently shifts earlier in the development lifecycle. Traditional software engineering adopted shift-left practices decades ago, embedding automated validation directly into development workflows. High-assurance industries such as aerospace followed a similar trajectory, prioritizing early verification to prevent costly downstream failures.

The 2025 exploit data indicates blockchain systems have reached a comparable inflection point. As value concentration grows, security controls applied only after development are no longer sufficient. **Correctness must be evaluated where logic is written, not after it has been deployed.**

Operationalizing shift-left security requires layered validation throughout the SDLC:

1	**Static analysis**	Identify logical flaws and unsafe patterns at development time.
2	**Unit testing**	Validate expected behavior under known conditions.
3	**Mutation testing**	Assess whether tests meaningfully enforce intended invariants.
4	**Fuzzing**	Explore unexpected execution paths and adversarial inputs.
5	**Formal verification**	Mathematically validate critical properties where required.

Large language models can play a valuable role in surfacing certain classes of issues and accelerating developer workflows. However, due to their probabilistic nature, LLM-based tools cannot serve as a sole source of security assurance.

In high-risk systems, security outcomes must be reproducible, measurable, and verifiable, not inferred from non-deterministic analysis alone.

As markets mature, security spend shifts left.

07

Deterministic vs. probabilistic security

Why reproducible findings matter

As smart contract systems continue to secure increasing amounts of economic value, the standard for security assurance must evolve. Not all security approaches offer the same level of confidence.

PROBABILISTIC

Relies on likelihood

These approaches, including pattern detection, heuristics, and many AI-driven tools, surface issues based on probability rather than certainty. While effective for triage and signal generation, their outputs are inherently non-deterministic and difficult to reproduce.

DETERMINISTIC

Decides by counterexample

It checks whether defined properties, invariants, and constraints hold on the paths it tests, and when one breaks, it returns the concrete counterexample that shows how. Outcomes are reproducible, measurable, and verifiable, qualities required when systems carry material financial and reputational risk.

This distinction has become increasingly important as artificial intelligence is applied to security workflows. In a 2025 open letter to the bank's suppliers, JPMorgan Chase's CISO Pat Opet warned that competition had pushed software providers to prioritize rapid feature development over robust security. The same pressure applies to AI security tooling, where an output that can't be reproduced is hard to hold to account.

The implication is not that AI lacks value, but that its role must be appropriately scoped. AI can accelerate discovery, expand surface coverage, and improve developer efficiency. However, it cannot replace a deterministic check that settles whether something is actually broken.

As blockchain systems mature, security assurance must shift from confidence inferred to evidence you can rerun.

In environments where failure carries irreversible financial consequences, a finding you can reproduce is the minimum standard.

About Olympix

The findings in this report highlight a consistent pattern: the majority of 2025 smart contract exploits were not caused by unknown vulnerabilities, but by logic flaws and untested assumptions introduced earlier in development.

Based on analysis of in-scope incidents, approximately 98% of exploits, representing roughly $247 million in losses, could have been prevented had vulnerabilities been identified prior to deployment.

Olympix was built to address this structural gap through a proactive, automated, and verifiable security model. Rather than relying on retrospective detection, the platform applies shift-left security principles to evaluate correctness as code is written, when risk can still be removed, not merely monitored.

This approach reflects practices long established in mature industries such as traditional software and aerospace, where early, deterministic validation is used to prevent high-impact failures downstream. By introducing reproducible and measurable security assurance earlier in the SDLC, Olympix supports a security posture centered on prevention, reproducible findings, and operational scalability.

TRUSTED BY

CIRCLE · SECURITIZE · aave · UNISWAP FOUNDATION

ARBITRUM FOUNDATION · Ethena · SkyMavis · LI.FI

crossmint · CORK · Fusion by IPOR · STAKE DAO

Yield.xyz · Uniform Labs · catalysis · Arc

Benchmark Olympix against your own audit

olympix.security/demo

Book a technical walkthrough