Introduction

A security review of the Unity protocol was done, focusing on the security aspects of the smart contracts. This audit was performed by 33Audits & Co with Samuel as the Security Researcher.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

Commit: d853d63f6662e174877954d97081df9f58df950a - Scope: Core contracts

About 33 Audits & Company

33Audits LLC is an independent smart contract security researcher company and development group. We conduct audits a as a group of independent auditors with various experience and backgrounds. We have conducted over 15 audits with dozens of vulnerabilities found and are experienced in building and auditing smart contracts. We have over 4 years of Smart Contract development with a focus in Solidity, Rust and Move. Check our previous work here or reach out on X @33audits.

About Unity

This audit is being performed on the core protocol contracts for Unity. The contracts in scope are the Unity core contracts.

Severity Definitions

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Findings Summary

ID	Title	Severity	Status

ID	Title	Severity	Status
[M-01]	Off-by-one interval math leads to extra allocation and permanent schedule drift	Medium	Acknowledged

Medium

[M-01] Off-by-one interval math leads to extra allocation and permanent schedule drift

Description

The first interval update for each flow (PLSX burn, PLS burn, PLSX build, PLS build) over-allocates by exactly one full interval. On the initial call, `missedIntervals = floor(timeElapsed / INTERVAL_TIME)`. However, the implementation:

- sets `_lastIntervalNumber` to `last + missedIntervals + 1`; and
- computes `_totalAmountForInterval` as `amountPerInterval + missedIntervals * amountPerInterval` (i.e., `(missedIntervals + 1) * amountPerInterval`).

Additionally, on subsequent (non-first) calls the updater increments the local `_missedIntervals` again (e.g., `_missedIntervals++`), compounding the skew. In effect, the code treats the currently in-progress interval as fully elapsed and advances counters by one. This permanently shifts the schedule forward and over-allocates funds on the first update of each flow.

Impact

- **Medium Impact**: One extra interval allocated on first update for each flow, affecting allocation accuracy
- **Medium Likelihood**: Occurs on every first update, causing permanent schedule drift

Recommendations

- Allocate only fully elapsed intervals; remove the implicit +1:
  - Set `_lastIntervalNumber = last + missedIntervals`.
  - Set `_totalAmountForInterval = amountPerInterval * missedIntervals` (then cap to available balance).
- Update interval updaters consistently:
  - Remove the extra `_missedIntervals++`.
  - If `missedIntervals == 0`, skip writing a new interval.
  - Advance `last...StartTimestamp` by `missedIntervals * INTERVAL_TIME`.
- Apply the above to all flows:
  - PLSX burn, PLS burn, PLSX build, PLS build (TitanX/ETH × Burn/Build).
- Alternative mitigation (if math is not changed):
  - Initialize the first start timestamp 10 minutes earlier (or set `startTimeStamp` so the first 10-minute slot is already complete), avoiding the first-call over-allocation.

Status

Acknowledged - Developer acknowledged the issue, we agreed that the protocol could mitigate the issue by setting the initial time to 10 minutes in constructor to bypass the first 10-minute interval, effectively avoiding the extra allocation issue.

Conclusion

This security audit of the Unity protocol identified one medium severity issue related to interval math and allocation timing. The development team has acknowledged the finding and implemented an alternative solution by setting the initial time to 10 minutes in the constructor, effectively bypassing the first interval to avoid the extra allocation issue. Additional findings were considered design choices and should be monitored. The protocol demonstrates good security practices overall with minimal vulnerabilities identified.

Timeline

- **Audit Period**: [08/04/2025 - 08/08/2025]
- **Report Delivery**: [08/21/2025]

This report was prepared by 33Audits & Co and represents our independent security assessment of the Unity protocol smart contracts.