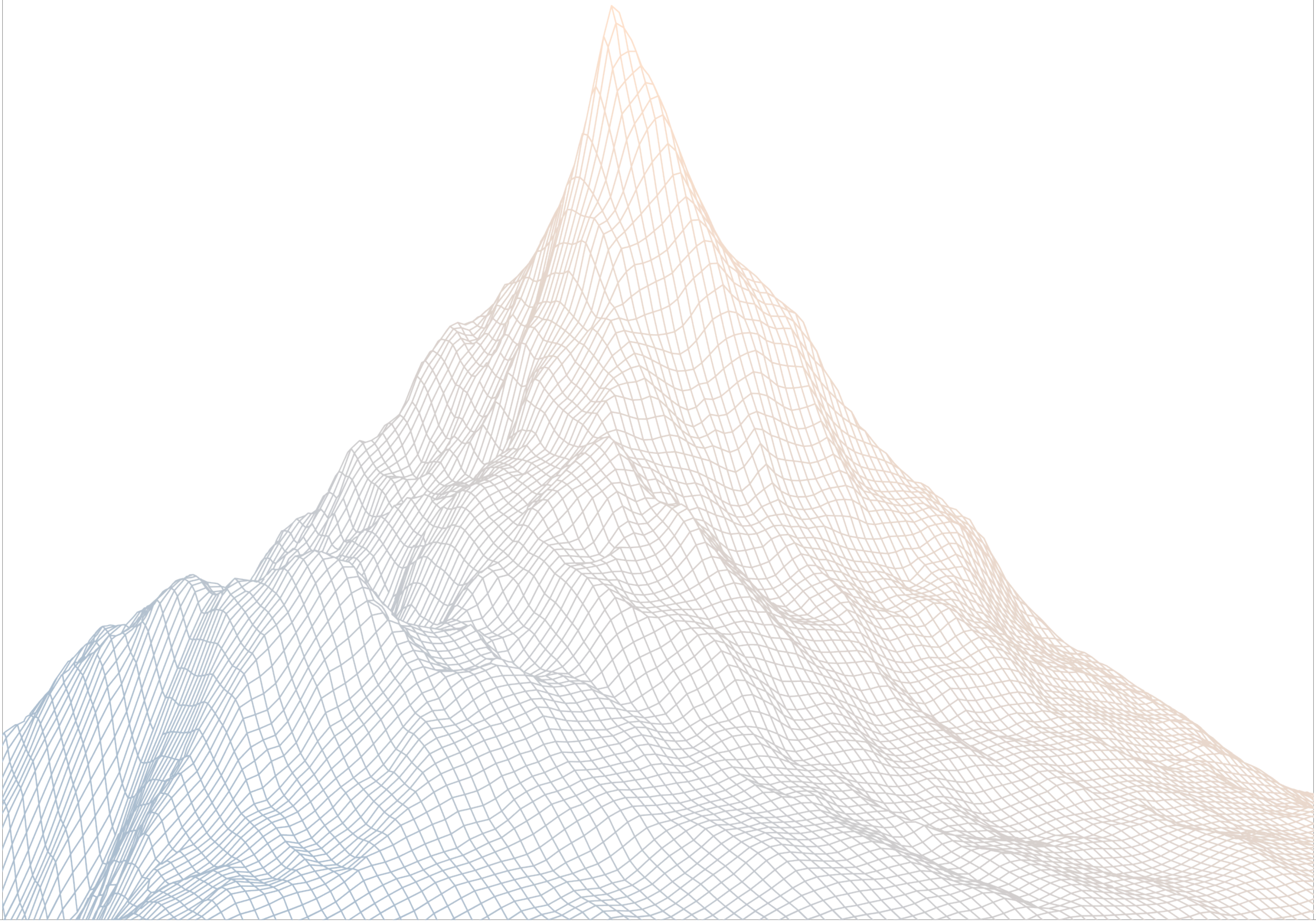
Zenith

Ignition

Smart Contract
Security Assessment

VERSION 1.1



AUDIT DATES: January 13th to January 15th, 2026

AUDITED BY: Matte
peakbolt

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About Ignition

Ignition is a liquid staking protocol built on Fogo, a high-performance SVM Layer 1 blockchain optimized for ultra-low latency onchain trading. Users can stake their FOGO tokens to receive iFOGO, a liquid staking token that maintains staking rewards while enabling seamless integration into DeFi applications

2.2 Scope

The engagement involved a review of the following targets:

Target	spl-stake-pool
Repository	https://github.com/Tempest-Finance/spl-stake-pool
Commit Hash	`544c269ce395a65e4615f2153b51c8e8dfd14213`
Files	`Changes in PR-22`

2.3 Audit Timeline

January 13, 2026	Audit start
January 15, 2026	Audit end
January 28, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	0
Low Risk	0
Informational	0
Total Issues	**1**

3 Findings Summary

ID	Description	Status
H-1	WithdrawStakeWithSession burns fewer tokens than authorized due to recalculation	Resolved

4 Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] WithdrawStakeWithSession burns fewer tokens than authorized due to recalculation

SEVERITY: High	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- program/src/processor.rs#L3458-L3502

Description:

Withdrawals initiated through `WithdrawStakeWithSession` are meant to burn the exact pool tokens the user supplies, leaving pool accounting unchanged except for the removed lamports. In the session branch, the code recomputes the burn amount from lamports instead of reusing the already-approved `pool_tokens_burnt`, so the intended invariant “lamports removed ÷ current exchange rate = tokens burned” is broken whenever the lamports/token ratio is non-integer. That mismatch lets users exit with more lamports per token than they provided.

The flow is:

1. tokens and fees are processed up front, deriving `pool_tokens_burnt` and `withdraw_lamports`;
2. a stake account is split for `split_lamports` (withdraw minus rent if the user funded rent separately);
3. inside the session-only branch, the burn amount is recalculated with a deposit-style converter that floors the result. A narrow snippet shows the incorrect recomputation:

- `calc_pool_tokens_for_deposit(split_lamports …)` // floors to the next-lower token count

Because `calc_pool_tokens_for_deposit` performs `lamports * pool_supply / total_lamports`, any prior rounding down of `withdraw_lamports` causes a second downward rounding. Example: with `total_lamports = 10,000,000` and `pool_token_supply = 9,000,000`, a 4,500,001-token withdrawal yields `withdraw_lamports`

`= 5,000,001`. The session path re-derives a burn of `5,000,001 * 9,000,000 / 10,000,000 = 4,500,000`, one token less than the user committed, yet the pool still hands over 5,000,001 lamports. Pool token supply shrinks too little while lamports drop fully, diluting token value for everyone else.

The bug propagates further: fee math now uses an inconsistent token base, and slippage checks were done against the larger, original `pool_tokens_burnt` so the actual burn can fall outside the caller's intended price bounds. If a third-party payer covered rent, the recalculation also burns tokens against that rent without reimbursing the payer, shifting value between participants unexpectedly.

Recommendations:

Reuse the original `pool_tokens_burnt` for the session burn, just as the non-session path does, and avoid deposit-style rounding during withdrawal. A minimal fix is to remove the recomputation and burn the precomputed amount:

```diff
- let pool_tokens_burnt = stake_pool
-     .calc_pool_tokens_for_deposit(if is_user_payer {
-       split_lamports
-     } else {
-       split_lamports.saturating_add(required_rent)
-     })
-     .ok_or(StakePoolError::CalculationFailure)?;
+ // Burn exactly the tokens authorized for this withdrawal
+ let pool_tokens_burnt = pool_tokens_burnt;
```

If rent must be fronted by the program signer, pre-fund it and settle separately rather than altering the burn calculation.

Ignition: Resolved with PR-51.

Zenith: Verified. Resolved by funding `stake_split_to` rent using reserve stake account and keep `pool_tokens_burnt` unchanged.