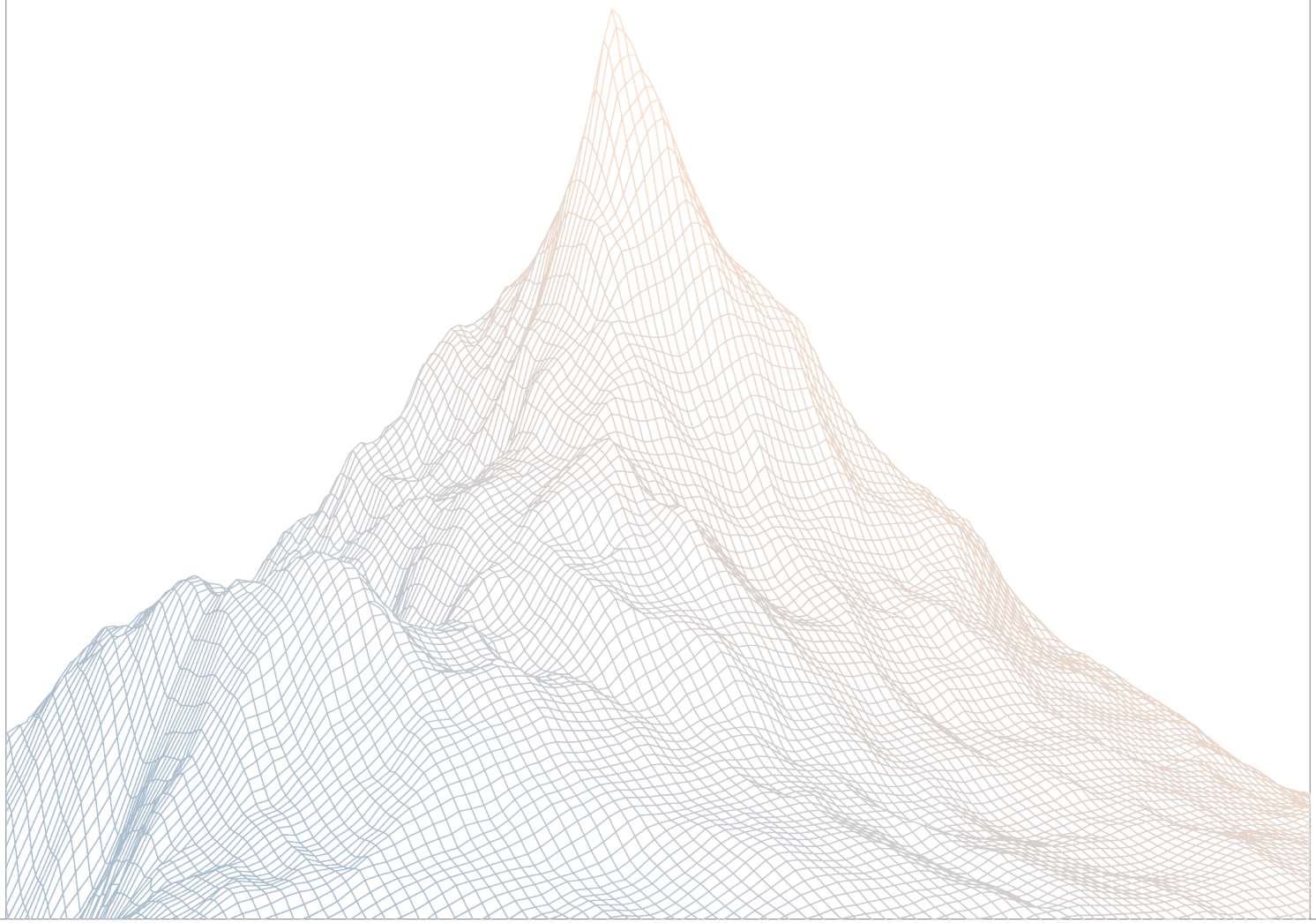
Zenith

Ignition

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: December 23rd to December 29th, 2025

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	`3bf58d156182dadf310546355fd9ba6b9ed1a2dd`
Files	`Diff from 8f705b3cb1a80e49ef4037a593d10713499ea8d7`

2.3 Audit Timeline

December 23, 2025	Audit start
December 29, 2025	Audit end
January 7, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	2
Informational	1
Total Issues	**5**

3

Findings Summary

ID	Description	Status
M-1	Mutable payer in deposit_wsol_with_session() is incorrectly marked readonly	Resolved
M-2	WSOL deposit and withdrawal will always fail for Token2022 stake pools	Acknowledged
L-1	Missing slippage parameters for WSOL session deposit/withdrawal	Resolved
L-2	Missing validation of burn_from_pool_info authority in process_withdraw_wsol_with_session	Resolved
I-1	rent_lamports calculation in process_deposit_wsol_with_session() can be simplified	Resolved

4

Findings

4.1 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Mutable payer in `deposit_wsol_with_session()` is incorrectly marked readonly

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

Target

- instruction.rs#L2725

Description:

`deposit_wsol_with_session()` is intended to initialize (if needed) `transient_wsol_account` during the WSOL deposit flow, with a `payer` funding the required rent.

However, the instruction marks `payer` as `AccountMeta::new_readonly(*payer, true)` instead of mutable, preventing the payer's lamports from being debited to create/fund the transient account.

This causes deposits to revert whenever `transient_wsol_account` is not already initialized, causing `deposit_wsol_with_session()` to fail.

Recommendations:

Mark `payer` as mutable `AccountMeta::new(*payer, true)` .

Ignition: Resolved with `@ae2d3df019 ...`

Zenith: Verified.

[M-2] WSOL deposit and withdrawal will always fail for Token2022 stake pools

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- processor.rs#L2584
- processor.rs#L3257

Description:

`process_deposit_wsol_with_session()` / `process_withdraw_wsol_with_session()` are intended to let users deposit/withdraw via the wSOL path when using a session signer.

However, they enforce `wsol_mint_info.owner == *token_program_info.key`, implicitly assuming `token_program_info` is the SPL Token program, which is incompatible when the StakePool is configured to use Token-2022. The same issue applies to use of `token_program_info` in validating `wsol_token_info` and WSOL transfers.

This causes the wSOL deposit/withdraw path to always fail (DoS) for Token-2022 pools, preventing users from using the session-based wSOL flow.

Recommendations:

In `process_deposit_wsol_with_session()` / `process_withdraw_wsol_with_session()` , add a separate SPL-Token program account for WSOL related actions. And it validate against `spl_token::id()` using `spl_token::check_program_account()`.

Ignition: Acknowledged. Known fogo-sessions-sdk limitation — Token2022 is not supported.

4.2 Low Risk

A total of 2 low risk findings were identified.

[L-1] Missing slippage parameters for WSOL session deposit/withdrawal

SEVERITY: Low	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- processor.rs#L4033-L4040

Description:

`deposit_wsol_with_session()` and `withdraw_wsol_with_session()` are intended to let users deposit/withdraw using wSOL via a session-based flow.

However, they do not expose a slippage protection parameters for minimum expected output pool tokens or SOL.

This makes the session flow vulnerable to unfavorable output, causing users to receive materially less value than expected.

That could happen when a malicious manager sandwich the deposit/withdraw TXs with fee increase/decrease and steal from the outputs. Or it could occur when stake account are drained during a hack.

Recommendations:

Consider adding slippage protection parameters for `deposit_wsol_with_session()` and `withdraw_wsol_with_session()` to allow users to decide whether to provide an minimum received tokens.

Ignition: Resolved with `@b4a65d6028 ...`

Zenith: Verified. Resolved by adding slippage protection parameters.

[L-2] Missing validation of `burn_from_pool_info` authority in `process_withdraw_wsol_with_session`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- processor.rs#L3249

Description:

`process_withdraw_wsol_with_session` is intended to burn pool tokens from `burn_from_pool_info` as part of a user's withdraw flow.

However, it does not validate that `user_pubkey` is the token authority/owner of `burn_from_pool_info`.

This allows one to use pool token account that is owned by the session instead as the transfer/burn are signed by the session key. Though the session is unlikely to have any pool tokens, it is recommended to verify that the session is only interacting with the user's accounts.

Recommendations:

In `process_withdraw_wsol_with_session`, verify that `burn_from_pool_info`'s `authority` matches `user_pubkey`.

Ignition: Resolved with `@74a52adac5 ...`

Zenith: Verified.

4.3 Informational

A total of 1 informational findings were identified.

[I-1] `rent_lamports` calculation in `process_deposit_wsol_with_session()` can be simplified

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- processor.rs#L2704-L2712

Description:

In `process_deposit_wsol_with_session()`, the `rent_lamports` calculation can be simplified to just `rent.minimum_balance(spl_token::state::Account::LEN)`.

That is because `wsol_transient_info.lamports() == 0` since `close_account()` will send all the SOL to `program_signer_info`.

```
// Refund rent to payer
let rent_lamports = rent
    .minimum_balance(spl_token::state::Account::LEN)
    .max(1)
    .saturating_sub(if wsol_transient_info.lamports() > 0 {
        wsol_transient_info.lamports()
    } else {
        0
    });
```

Ignition: Resolved with @b4a65d6028 ...

Zenith: Verified. Resolved by simplifying `rent_lamports`calculation.