Security Audit - Fogo Stake Pool

conducted by Neodyme AG

Lead Auditor:	Sebastian Fritsch
Second Auditor:	Nico Gründel
Administrative Lead:	Jasper Slusallek

October 24, 2025

Table of Contents

Appendices

1 | Executive Summary

Neodyme audited **FirstSet's** on-chain Stake Pool program fork during October 2025. The stake pool will be deployed on the Fogo Solana fork and will support Fogo-specific features.

The audit was performed as a differential audit between the official Anza Stake Pool and the fork by FirstSet. According to Neodymes Rating Classification, **1 security relevant** and **1 informational** were found. The number of findings identified throughout the audit, grouped by severity, can be seen in Figure 1.

Figure 1: Overview of Findings

The auditors reported all findings to the FirstSet developers, who addressed them promptly. The security fixes were verified for completeness by Neodyme. In addition to these findings, Neodyme delivered the FirstSet team a list of nit-picks and additional notes that are not part of this report.

2 | Introduction

During October 2025, FirstSet engaged Neodyme to do a detailed security analysis of their Fogo Stake Pool. Two senior security researchers from Neodyme conducted independent audits of the contract between the 17th of October 2025 and the 24th of October 2025. Both auditors have a long track record of finding critical and other vulnerabilities in Solana programs, as well as in Solana's core code itself, and have extensive knowledge about the stake pool program.

The audit focused on the contract's technical security. In the following sections, we present the new features developed by FirstSet from a technical perspective and will explain our findings and provide some general notes for considerations that may be useful in the future.

Summary of Findings

All found issues were **quickly remediated**. In total, the audit revealed:

0 critical • **0** high-severity • **1** medium-severity • **0** low-severity • **1** informational

issues. We further discuss all authorities in Section 4.3.

3 | Scope

The contract audit's scope comprised:

- a thorough check of all changes in the forked version of the SPL stake pool,
- a review of the correct usage of Fogo sessions

Neodyme considers the source code, located at https://github.com/Firstset/stake-pool-v2/tree/wsol-adaptor, in scope for this audit. Third-party dependencies are not in scope. FirstSet only relies on the original stake pool program and the Fogo SDK. During the audit, minor changes and fixes were made by FirstSet, which the auditors also reviewed in-depth.

Relevant source code revisions are:

- `2bd399877ba81b7555a9f9f3a9a9b4ae858f7e18` · Start of the audit
- `ff887c769f6746f05d55e8250f328252a7c9f8cc` · Last reviewed revision

4 | Project Overview

This section briefly outlines Fogo Stake Pool's functionality, design, and architecture, followed by a detailed discussion of all related authorities.

Functionality

The FirstSet Stake Pool provides the same core functionality as the SPL Stake Pool (see https://github.com/solana-program/stake-pool
). For more details, refer to the SPL Stake Pool audits at https://github.com/anza-xyz/security-audits/tree/master/spl. In addition, FirstSet introduced two new instructions enabling wSOL deposits and withdrawals. This feature allows integration with Fogo Sessions, which lets users delegate control of their funds to Fogo, enabling spending without submitting transactions themselves. The new deposit instruction unwraps wSOL before following the standard SPL deposit path. Similarly, the new withdraw instruction rewraps SOL into wSOL after withdrawal.

Instructions

In addition to the existing instructions, two new instructions were added:

Instruction	Category	Summary
DepositWsolWithSession	Permissionless	Unwrap wSOL into SOL and deposit them into the stake pool in return for pool tokens
WithdrawWsolWithSession	Permissionless	Burn pool tokens in exchange for stake pool SOL and wrap them into wSOL tokens

Authority Structure and Off-Chain Components

There are several authorities present, which we will detail in the following sections.

Stake Pool owner

The pool manager of a stake pool is responsible for delegating deposited funds and managing the set of validators in the pool. Therefore, the manager can control the resulting yield but cannot withdraw or burn user funds.

Fogo Paymaster

Fogo Sessions allow a user to delegate control over their funds to a centralized Fogo Paymaster. Users are able to define the scope of control with respect to allowed programs, amounts and time. In the case of a compromise of the Paymaster infrastructure, those funds are at risk.

Upgrade Authority

FirstSet plans to use a 3/5 Squads multisig for their update authority, with two people from the FirstSet team and the remaining three from the Fogo ecosystem. As the main program was not deployed at the time of the audit, Neodyme has not verified this.

5 | Findings

This section outlines all of our findings. They are classified into one of five severity levels, detailed in Appendix C. In addition to these findings, Neodyme delivered the FirstSet team a list of nit-picks and additional notes which are not part of this report.

All findings are listed in Table 2 and further described in the following sections.

Identifier	Name	Severity	Status
ND-FSP-MD-01	Creation of transient wSOL account can be blocked	MEDIUM	Resolved
ND-FSP-IN-01	Stake pool upgrades	INFORMATIONAL	Resolved

Table 2: Findings

[ND-FSP-MD-01] Creation of transient wSOL account can be blocked

Severity	Impact	Affected Component	Status
MEDIUM	DoS	wSOL Deposits	Resolved

In the new `DepositWsolWithSession` instruction, FirstSet transfers wSOL from the user to a transient, program-owned wSOL account, which is later closed to unwrap the SOL. It creates this transient account using the system program's `CreateAccount` instruction at the PDA `["transient_wsol", user_pubkey.as_ref()]`. However, `CreateAccount` fails if the target account already holds a nonzero balance of lamports. Since the transient account address is predictable, an attacker could pre-fund it, effectively blocking wSOL deposits for that user.

processor.rs, lines 2674-2694

```
// Derive the *expected* PDA for
// the transient WSOL account
let (expected_transient_pda, transient_bump) = Pubkey::find_program_address(
    &[b"transient_wsol", user_pubkey.as_ref()],
    program_id,
);

if expected_transient_pda != *transient_wsol_info.key {
    msg!("transient_wsol_account does not match PDA derived from seeds");
    return Err(ProgramError::InvalidSeeds);
}

// Create a temporary WSOL account for the session
let rent = Rent::get()?;
let rent_lamports = rent.minimum_balance(spl_token::state::Account::LEN);
let create_ix = solana_program::system_instruction::create_account(
    fee_payer_info.key,                    // payer (wallet or session key)
    transient_wsol_info.key,               // new account address (PDA)
    rent_lamports,                         // rent-exempt lamports
    spl_token::state::Account::LEN as u64, // space for a token account
    token_program_info.key,                // OWNER **must** be SPL-Token!
);
```

Resolution

We recommend using the idempotent account creation mechanism found in the associated token program at https://github.com/solana-program/associated-token-account/blob/main/program/src/tools/account.rs#L16. FirstSet quickly deployed a fix in commit `d147e71adeab8378dc5f51442f92fae1be9dbd26`. Neodyme verified the fix.

[ND-FSP-IN-01] Stake pool upgrades

Severity	Impact	Affected Component	Status
INFORMATIONAL	LoF	Balance tracking	Resolved

FirstSet forked off from the SPL stake pool before the fixes for a rare LoF bug were published. We added them to the coordinated disclosure process, and FirstSet quickly updated its program. As the Mainnet program was not launched yet, no user funds were at risk at any time.

Resolution

FirstSet applied the patches in PR 19. Neodyme verified the fixes.

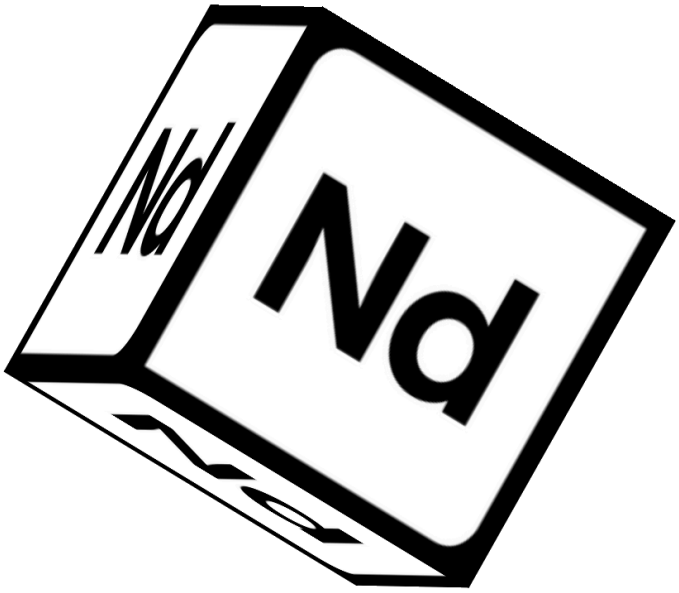
A | About Neodyme

Security is difficult.

To understand and break complex things, you need a certain type of people. People who thrive in complexity, who love to play around with code, and who don't stop exploring until they fully understand every aspect of it. That's us.

Our team never outsources audits. Having found over 80 High or Critical bugs in Solana's core code itself, we believe that Neodyme hosts the most qualified auditors for Solana programs. We've also found and disclosed critical vulnerabilities in many of Solana's top projects and have responsibly disclosed issues that could have resulted in the theft of over $10B in TVL on the Solana blockchain.

All of our team members have a background in competitive hacking. During such hacking competitions, called CTFs, we competed and collaborated while finding vulnerabilities, breaking encryption, reverse engineering complicated algorithms, and much more. Through the years, many of our team members have won national and international hacking competitions, and keep ranking highly among some of the hardest CTF events worldwide. In 2020, some of our members started experimenting with validators and became active members in the early Solana community. With the prospect of an interesting technical challenge and bug bounties, they quickly encouraged others from our CTF team to look for security issues in Solana. The result was so successful that after reporting several bugs, in 2021, the Solana Foundation contracted us for source code auditing. As a result, Neodyme was born.



B | Methodology

We are not checklist auditors.

In fact, we pride ourselves on that. We adapt our approach to each audit, investing considerable time into understanding the program upfront and exploring its expected behavior, edge cases, invariants, and ways in which the latter could be violated.

We use our uniquely deep knowledge of Solana internals, and our years-long experience in auditing Solana programs to find bugs that others miss. We often extend our audit to cover off-chain components in order to see how users could be tricked or the contract affected by bugs in those components.

Nonetheless, we also have a list of common vulnerability classes, which we always exhaustively look for. We provide a sample of this list here.

Select Common Vulnerabilities

Our most common findings are still specific to Solana itself. Among these are vulnerabilities such as the ones listed below:

- Insufficient validation, such as:
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Account confusions
  - Missing freeze authority checks
  - Insufficient SPL account verification
  - Dangerous user-controlled bumps
  - Insufficient Anchor account linkage
- Account reinitialization vulnerabilities
- Account creation DoS
- Redeployment with cross-instance confusion
- Missing rent exemption assertion
- Casting truncation
- Arithmetic over- or underflows
- Numerical precision and rounding errors
- Anchor pitfalls, such as accounts not being reloaded
- Non-unique seeds
- Issues arising from CPI recursion
- Log truncation vulnerabilities
- Vulnerabilities specific to integration of Token Extensions, for example unexpected external token hook calls

Apart from such Solana-specific findings, some of the most common vulnerabilities relate to the general logical structure of the contract. Specifically, such findings may be:

- Errors in business logic
- Mismatches between contract logic and project specifications
- General denial-of-service attacks
- Sybil attacks
- Incorrect usage of on-chain randomness
- Contract-specific low-level vulnerabilities, such as incorrect account memory management
- Vulnerability to economic attacks
- Allowing front-running or sandwiching attacks

Miscellaneous other findings are also routinely checked for, among them:

- Unsafe design decisions that might lead to vulnerabilities being introduced in the future
    - Additionally, any findings related to code consistency and cleanliness
- Rug pull mechanisms or hidden backdoors

Often, we also examine the authority structure of a contract, investigating their security as well as the impact on contract operations should they be compromised.

Over the years, we have found hundreds of high and critical severity findings, many of which are highly nontrivial and do not fall into the strict categories above. This is why our approach has always gone way beyond simply checking for common vulnerabilities. We believe that the only way to truly secure a program is a deep and tailored exploration that covers all aspects of a program, from small low-level bugs to complex logical vulnerabilities.

C | Vulnerability Severity Rating

We use the following guideline to classify the severity of vulnerabilities. Note that we assess each vulnerability on an individual basis and may deviate from these guidelines in cases where it is well-founded. In such cases, we always provide an explanation.

Severity	Description
CRITICAL	Vulnerabilities that will likely cause loss of funds. An attacker can trigger them with little or no preparation, or they are expected to happen accidentally. Effects are difficult to undo after they are detected.
HIGH	Bugs that can be used to set up loss of funds in a more limited capacity, or to render the contract unusable.
MEDIUM	Bugs that do not cause direct loss of funds but that may lead to other exploitable mechanisms, or that could be exploited to render the contract partially unusable.
LOW	Bugs that do not have a significant immediate impact and could be fixed easily after detection.
INFORMATIONAL	Bugs or inconsistencies that have little to no security impact, but are still noteworthy.

Additionally, we often provide the client with a list of nit-picks, i.e. findings whose severity lies below Informational. In general, these findings are not part of the report.

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
Germany

E-Mail: contact@neodyme.io

https://neodyme.io