Ignition Fogo Locker

SECURITY ASSESSMENT REPORT

February 9, 2026

Prepared for:

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

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

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the security assessment conducted by CODESPECT for Ignition Fogo Locker update of the Jupiter Locker program. The Locker program provides an escrow and token time-based distribution capabilities.

This audit focuses on the Fogo Locker program update on top of Jupiter Locker, which introduced the Fogo session based authorization to the escrow users.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of smart contracts, execution testing.

CODESPECT found three points of attention, one classified as `Low`, one classified as `Info` and one classified as `Best Practice`. All of the issues are summarised in Table 2. **Organisation of the document is as follows:**

- **Section 5** summarizes the audit.
- **Section 6** describes the functionality of the code in scope.
- **Section 7** presents the issues.
- **Section 8** discusses the documentation provided by the client for this audit.
- **Section 9** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
Low	0	1	0
Informational	0	1	0
Best Practices	0	1	0
Total	**0**	**3**	**0**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	Fogo Locker
Type of Project	Escrow
Duration of Engagement	3 Days
Duration of Fix Review Phase	1 Day
Draft Report	February 9, 2026
Final Report	February 9, 2026
Repository	fogo-locker
Commit (Audit)	6822157c99253244f330a1fbb27541e31dfc9013
Commit (Final)	484c05d6c82c85fc3977060150f0a4307d2ea1de
Documentation Assessment	Low
Test Suite Assessment	Low
Auditors	jecikPo, Bount3yHunt3r

Table 3: Summary of the Audit

5.1 Scope - Audited Files

1	locker/src/instructions/escrow_instructions/create_vesting_escrow_with_session.rs	154
2	locker/src/instructions/escrow_instructions/claim_with_session.rs	106
3	locker/src/util/token.rs	65
4	locker/src/lib.rs	26
5	locker/src/errors.rs	6
6	locker/src/instructions/escrow_instructions/mod.rs	5
7	locker/src/util/token2022.rs	1
	Total	**363**

6 Summary of Issues

	Finding	Severity	Update
1	Extra guardrails in case compromised session key for `create_vesting_escrow_with_session`	Low	Fixed
2	Session version adds missing validations from original instructions	Info	Fixed
3	Lack of correct session validation	Best Practices	Fixed

7 System Overview

The **Fogo Locker** program is a Solana-based token vesting escrow system. It allows creators to lock tokens into escrow accounts that vest over time to designated recipients. This audit scope covers two new instructions added to the Locker program: `create_vesting_escrow_with_session` and `claim_with_session`. These instructions integrate with the external **Fogo Sessions** framework to enable delegated signing, allowing authorized frontend applications to submit transactions on behalf of users without requiring a wallet signature for each operation.

Fogo Sessions Integration

Fogo Sessions is an external framework that enables ephemeral, scoped delegation of transaction signing. A session is created as a **keypair account** (not a PDA) whose private key is held client-side by the application. During session creation (`start_session` on the Session Manager program), the user authorizes:

- A set of **authorized programs** that the session may interact with (resolved via a domain registry)
- A set of **authorized token mints** with specific amounts, or all tokens
- An **expiration** timestamp

As part of session creation, the Session Manager program approves the session keypair as a **delegate** on the user's token accounts via SPL `ApproveChecked`, granting it the ability to transfer tokens up to the approved amount.

Both new Locker instructions use `Session::extract_user_from_signer_or_session()` from the Fogo Sessions SDK. This function accepts either a direct wallet signer or a session account and returns the underlying user's public key. The instructions then validate that the extracted user is authorised to perform the requested operation.

For session-based token transfers, each consuming program derives a **program signer PDA** using a well-known seed (`PROGRAM_SIGNER_SEED`) from its own program ID. This PDA must co-sign the token transfer instruction via `invoke_signed`, serving as proof that the transfer was initiated through an authorised program rather than an arbitrary caller.

create_vesting_escrow_with_session

The `create_vesting_escrow_with_session` instruction creates a new token vesting escrow, functioning identically to the existing `create_vesting_escrow2` instruction but with session-based signing support. It works in the following way:

- The instruction accepts a `signer_or_session` account that must be a transaction signer. It is passed to `extract_user_from_signer_or_session()`, which validates it and returns the user's public key.
- The `sender_token` account is verified to be owned by the extracted user, ensuring the session can only transfer tokens belonging to the session's owner.
- The program derives the `program_signer` PDA using `PROGRAM_SIGNER_SEED` and the Locker program ID, and validates the passed account matches.
- The token mint is validated for supported extensions (Token-2022 support including transfer fees and transfer hooks).
- A new `VestingEscrow` account is initialised as a PDA seeded by a `base` keypair, storing the vesting parameters: start time, cliff time, frequency, cliff unlock amount, amount per period, number of periods, update recipient mode, and cancel mode.
- Tokens are transferred from the sender's token account to the escrow's associated token account using the Fogo Sessions SDK's `transfer_checked` instruction builder. The transfer is executed via `invoke_signed` with the program signer PDA, enabling the session-delegated transfer.
- Transfer Hook accounts are supported via remaining accounts for Token-2022 mints with the Transfer Hook extension.
- A `payer` account (separate from the session signer) covers the rent for the new escrow account.

claim_with_session

The `claim_with_session` instruction allows the vesting escrow recipient to claim unlocked tokens, functioning identically to the existing `claim2` instruction but with session-based signing support. It works in the following way:

- The `signer_or_session` account is validated via `extract_user_from_signer_or_session()` to extract the user's public key.
- The extracted user is verified to match the escrow's `recipient` field, ensuring only the designated recipient (or their authorised session) can claim.
- The `recipient_token` account is verified to be owned by the extracted user.

- The escrow must not have been cancelled (`cancelled_at == 0`).
- The escrow's `token_mint` is validated against the passed `token_mint` account via `has_one` constraint.
- The claimable amount is calculated based on the current timestamp, the vesting schedule parameters, and the previously claimed amount. The lesser of the claimable amount and the caller-specified `max_amount` is transferred.
- Tokens are transferred from the escrow's token account to the recipient's token account using the existing `transfer_to_user2` function, which uses `invoke_signed` with the escrow PDA as the signing authority.
- Memo transfer support is included for Token-2022 accounts that require incoming transfer memos.
- Transfer Hook accounts are supported via remaining accounts.

New Utility Code

The following utility additions support the session-based instructions:

- `transfer_to_escrow_with_session` (in `util/token.rs`) — Constructs a token transfer instruction using the Fogo Sessions SDK's `transfer_checked` builder. This sets the `signer_or_session` as the authority and the `program_signer` PDA as a co-signer. The instruction is executed via `invoke_signed` using the program signer's seeds. Transfer Hook extension accounts are supported via the remaining accounts.

Note that the `claim_with_session` instruction does **not** use a session-based transfer for the outgoing claim. Since the escrow PDA itself is the authority over the escrow token account, the claim transfer uses the standard `transfer_to_user2` function with `invoke_signed` using the escrow's PDA seeds. The session is only used to authenticate the caller's identity, not to authorise the token movement.

8 Issues

8.1 [Low] Extra guardrails in case compromised session key for `create_vesting_escrow_with_session`

File(s): `create_vesting_escrow_with_session.rs`

Description: The `create_vesting_escrow_with_sessions` instruction allows creation of an escrow using the Fogo session mechanism, where such a session has permissions to transfer the required token to the escrow's account. While this mechanism is secure as long as the session key is not compromised, it is recommended to add an extra guardrails in case that happens. In case the session key gets compromised a malicious user can create the escrow on the user's behalf and put himself as `recipient` of the escrow's vested token and effectively taking them from the user.

Impact: Possible to bypass session token usage limitations in case of session's key compromise.

Recommendation(s): To prevent such a scenario we recommend to use the session intent's `extra` field to store the `recipient` address (or potentially a set of possible addresses, depending on the use case). This `extra` field's contents could then be obtained from the session account and used to validate provided `recipient` address.

Status: Fixed

Update from Ignition: Fixed. Added `recipient == session_user` validation in `create_vesting_escrow_with_session`. The session account already contains the user's pubkey, so no extra field is needed. This prevents a compromised session key from creating escrows with an attacker-controlled recipient. The non-session instruction (`create_vesting_escrow_v2`) intentionally allows arbitrary recipients since it requires a direct wallet signature. 484c05d6c82c85fc3977060150f0a4307d2ea1de

Update from CODESPECT: This is fixed correctly too.

8.2 [Info] Session version adds missing validations from original instructions

File(s): `create_vesting_escrow_with_session.rs`

Description: The session version of `create_vesting_escrow` adds a validation that is missing in the original `create_vesting_escrow2.rs`:

```
constraint = sender_token.mint == token_mint.key() @ LockerError::InvalidEscrowTokenAddress
```

The original `CreateVestingEscrow2Ctx` does not validate that `sender_token.mint` matches `token_mint`.

Impact: This is an improvement in the session version. The original instruction should be updated to include this validation for consistency and security. If the token mint does not match, the transaction would fail.

Recommendation(s): Backport this validation to `create_vesting_escrow2.rs` for consistency across all escrow creation instructions.

Status: Fixed

Update from Ignition: Fixed. Added `sender_token.mint == token_mint.key()` constraint to `create_vesting_escrow2`, matching the validation already present in the session variant. 484c05d6c82c85fc3977060150f0a4307d2ea1de

Update from CODESPECT: This is fixed correctly.

8.3 [Best Practice] Lack of correct session validation

File(s): `create_vesting_escrow_with_session.rs` `claim_with_session.rs`

Description: Both new instructions `create_vesting_escrow_with_session` and `claim_with_session` are designed to handle only session based calls. They both obtain the `user_pubkey` using special function:

```
let user_pubkey = Session::extract_user_from_signer_or_session(
    &ctx.accounts.signer_or_session,
    ctx.program_id,
)
.map_err(|_| LockerError::InvalidSession)?;
```

However if the `signer_or_session` is not a session account, but just a normal `Signer` it will not fail. As per the `fogo-sessions-sdk` code, this is the only condition for failure:

```
if !info.is_signer {
    return Err(SessionError::MissingRequiredSignature);
}
```

which will never be hit as Anchor requires the account to be a `Signer`.

Impact: The Session-only instructions allow non-session calls.

Recommendation(s): is to add `is_session` function to validate it correctly:

```
require!(     is_session(&ctx.accounts.signer_or_session),
    LockerError::InvalidSession
);
```

Status: Fixed

Update from Ignition: Fixed. Added `is_session()` validation to both `create_vesting_escrow_with_session` and `claim_with_session` instructions. 484c05d6c82c85fc3977060150f0a4307d2ea1de

Update from CODESPECT: This is fixed correctly.

9 Evaluation of Provided Documentation

The **Ignition Fogo Locker** documentation was provided in the form of a README, NatSpec and git comments:

- **NatSpec:** The in-code NatSpec comments were rather basic, yet sufficient and generally helpful in explaining specific flows and code branches.
- **README.md:** The provided file offered just a build and run explanation, but not architecture, design assumptions and flows.

Considering the fact that the fogo session update to an existing and established and audited program, was rather a simple change, the lacks of documentation were not that impacting, however **CODESPECT** strongly recommends updating it with the new Fogo session features.

10 Test Suite Evaluation

10.1 Compilation Output

```
% anchor build
[...]
   Finished `test` profile [unoptimized + debuginfo] target(s) in 17.44s
     Running unittests src/lib.rs (fogo-locker/target/debug/deps/locker-210c6bc85468e2c2)
```

10.2 Tests Output

Skipping the output as no tests related to newly added functionality were present. The test suite contained only tests of the existing instructions.

10.3 Notes on the Test Suite

CODESPECT strongly suggest to add tests covering the fogo session functionality, especially the negative tests related to handling invalid or malicious session related accounts.