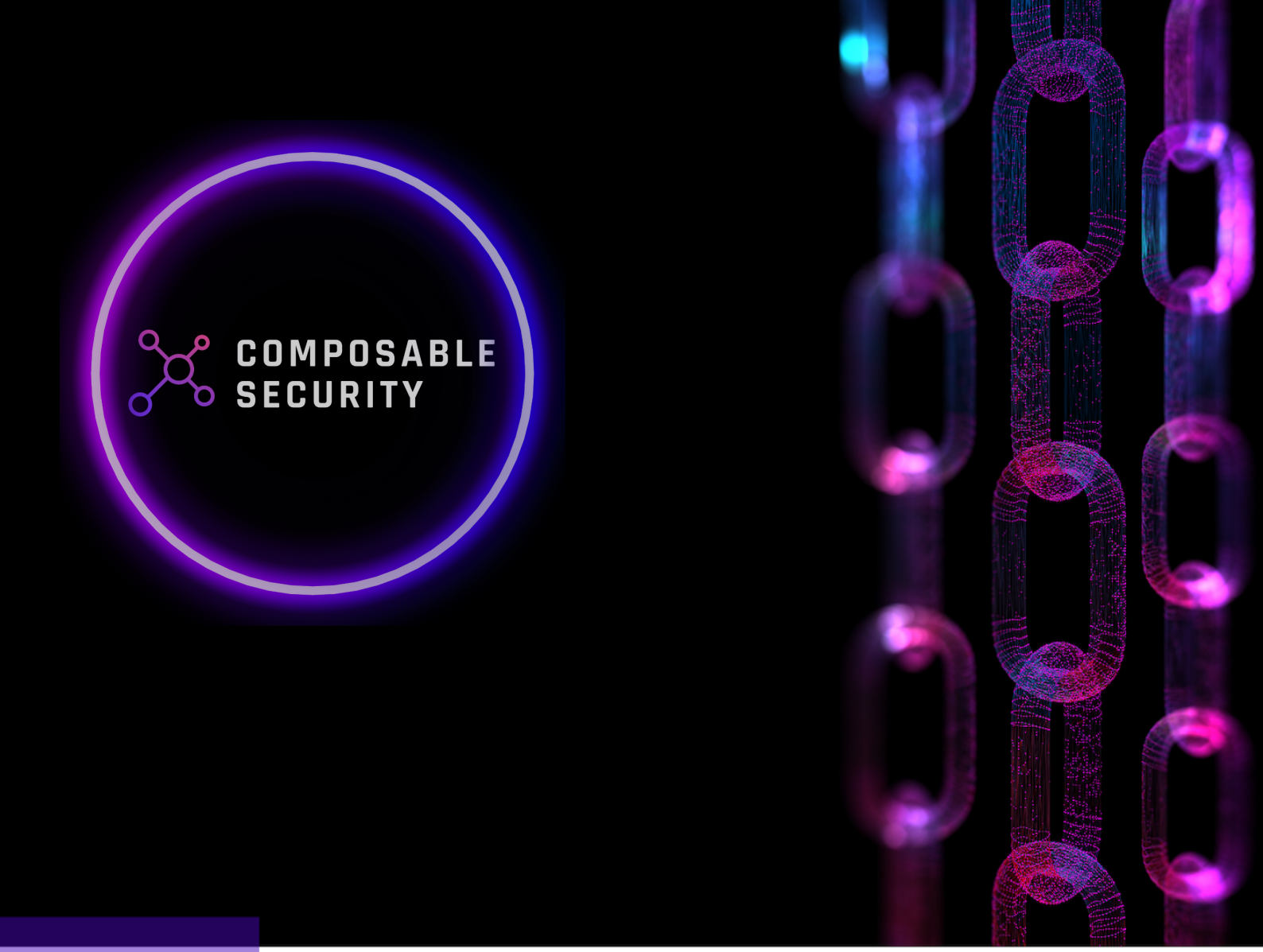


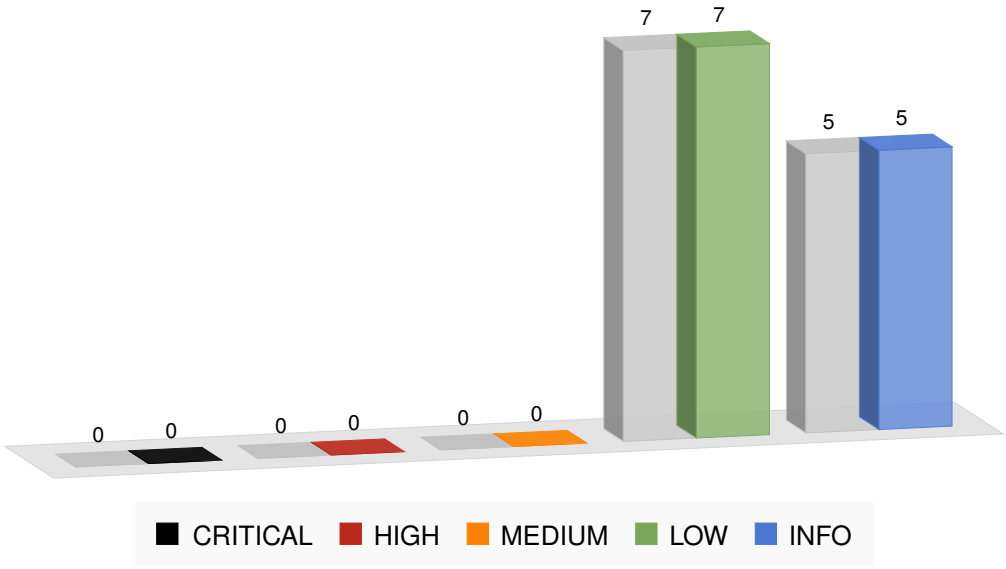
REPORT

Smart contract security review for Filecoin

Prepared by: Composable Security
Report ID: FIL-1132b525
Test time period: 2026-01-19 - 2026-01-30
Retest time period: 2026-02-16 - 2026-02-16
Report date: 2026-02-16
Version: 1.1
Visit: composable-security.com

Contents

1. Retest summary (2026-02-16)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team participated in a one-time iteration to verify if the vulnerabilities detected during the tests (between 2026-01-19 and 2026-01-30) were correctly removed and no longer appear in the code.

The current status of detected issues is as follows:

- The FilOz team decided to postpone the implementation of the fixes and recommendations due to project timelines and resource constraints.
- The Filecoin Pay project has been already released and due to non-critical issues it has been decided not to update it yet.
- The fixes and recommendations are planned to be implemented in the next release.
- The issue from FIL-1132b525-L01 has been described in the "Known Issues" section in the documentation.

1.2. Scope

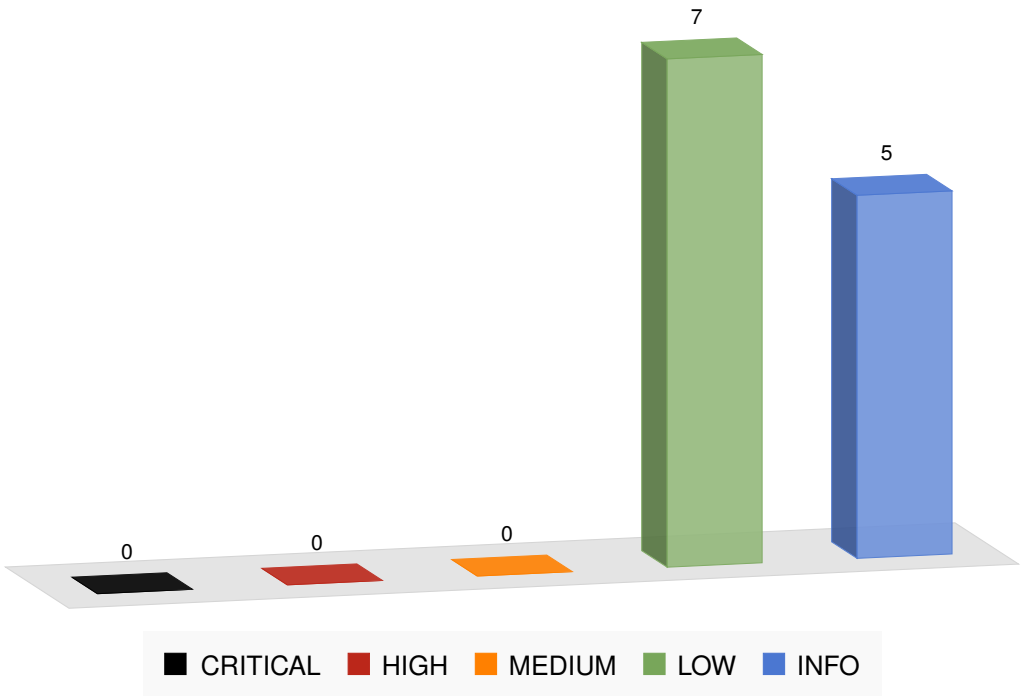
The retest scope included the same contracts, on a different commit in the same repository.

GitHub repository: https://github.com/FilOzone/filecoin-pay
CommitID: c916dc5cd059c48ca5d7588416af9e6025fa1fc6

2. Current findings status

ID	Severity	Vulnerability	Status
FIL-1132b525-L01	LOW	Reduced operator's available lockupAllowance	ACKNOWLEDGED
FIL-1132b525-L02	LOW	Missing validation note	ACKNOWLEDGED
FIL-1132b525-L03	LOW	Blocking transactions using permit directly	ACKNOWLEDGED
FIL-1132b525-L04	LOW	Unauthorized deposit of WETH-like tokens	ACKNOWLEDGED
FIL-1132b525-L05	LOW	Fix validation check	ACKNOWLEDGED
FIL-1132b525-L06	LOW	Rate queue array never cleared	ACKNOWLEDGED
FIL-1132b525-L07	LOW	Invalid order of parameters in errors	ACKNOWLEDGED
ID	**Severity**	**Recommendation**	**Status**
FIL-1132b525-R01	INFO	Unused error definitions in Errors	ACKNOWLEDGED
FIL-1132b525-R02	INFO	Use custom errors instead of revert strings	ACKNOWLEDGED
FIL-1132b525-R03	INFO	Clear serviceFeeRecipient	ACKNOWLEDGED
FIL-1132b525-R04	INFO	Consider using newest and clearly specified Solidity version	ACKNOWLEDGED
FIL-1132b525-R05	INFO	Fix NatSpec comment mislabelling caller	ACKNOWLEDGED

3. Security review summary (2026-01-30)



3.1. Client project

The **Filecoin Pay** project is a sophisticated on-chain payment infrastructure designed for the Filecoin ecosystem. It enables automated, streaming payment flows between service providers and their clients using ERC20 tokens (and native FIL).

Core Purpose The protocol creates "payment rails" - configurable payment channels between a payer (client) and a payee (service provider). These rails support:

- Continuous rate-based payments - Streaming payments at a configurable tokens-per-epoch (block) rate
- One-time lump-sum payments - Immediate transfers from pre-locked funds
- Payment validation/arbitration - Optional third-party validator contracts that can modify, delay, or reject payments

3.2. Results

The **Filecoin** engaged Composable Security to review security of **Filecoin Pay**. Composable Security conducted this assessment over 2 person-week with 2 engineers.

The summary of findings is as follows:

- **No vulnerabilities with a high and medium impact on risk were identified.**
- 7 vulnerabilities with a **low** impact on risk were identified.
- 5 **recommendations** have been proposed that can improve overall security and help implement best practice.
- A lot of project risk is mitigated at the assumption stage. Roles that could adversely affect other parties-whether acting alone or together-are treated as trusted and governed outside the contract through separate agreements; for example, a validator role that both the payer and payee must agree to before it can be used.
- The team was engaged and the communication was very good.
- The project was well described and documented, making the review easier.

Composable Security recommends that **Filecoin** complete the following:

- Address all reported issues.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.
- Consider whether the detected vulnerabilities may exist in other places (or ongoing projects) that have not been detected during engagement.
- Review dependencies and upgrade to the latest versions.

3.3. Scope

The scope of the tests included selected contracts from the following repository.

GitHub repository: https://github.com/FilOzone/filecoin-pay
CommitID: 1132b525e623e5f459e520121627ed2a159d33c9

The detailed scope of tests can be found in Agreed scope of tests.

4. Project details

4.1. Projects goal

The Composable Security team focused during this security review on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **Filecoin** and their users.
- The secondary goal is to improve code clarity and optimize code where possible.

4.2. Agreed scope of tests

The subjects of the test were selected contracts from the **Filecoin** repository.

GitHub repository:
https://github.com/FilOzone/filecoin-pay
CommitID: 1132b525e623e5f459e520121627ed2a159d33c9

Files in scope:

```
.
└── src
    ├── Dutch.sol
    ├── Errors.sol
    ├── FilecoinPayV1.sol
    └── RateChangeQueue.sol
```

Documentation:

- **README.md** - Comprehensive user-facing documentation covering all key concepts (Account, Rail, Validator, Operator), core functions with detailed parameter descriptions, worked examples of the full deal lifecycle, and emergency scenarios/escape hatches.
- **SPEC.md** - Deep technical specification for implementers. Covers internal data structures, fundamental fund flow principles, lockup bucket mixing, eager invariant enforcement, and detailed explanations of account/rail settlement mechanics.
- **CHANGELOG.md** - Version history documenting deployments on Calibration and Mainnet, feature additions, breaking changes, and security fixes across versions 0.1.0 through 1.0.0 (M4: Filecoin Service Liftoff).

4.3. Threat analysis

This section summarizes the potential threats that were identified during the initial threat modeling performed before the security review. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Key assets that require protection:

- Users' funds in accounts.
- Locked funds and guaranties.
- Settlement integrity and lifcycle state.
- Operator approvals and limits.
- Rail configuration integrity.

Potential attackers goals:

- Theft of user's funds.
- Lock users' funds in the contract.
- Force or misroute payments.
- Privilege abuse/escalation.
- Denial of Service.
- Exploit signature/approval flows.

Potential scenarios to achieve the indicated attacker's goals:

- Influence or bypass the business logic of the system.
- Improper validation.
- Privilege escalation through incorrect access control to functions or badly written modifiers.
- Existence of known vulnerabilities (e.g., front-running, re-entrancy).
- Token quirks.
- Excessive power, too much in relation to the declared one.
- Permit misuse.
- Draining via settlement.
- Abusing rails configuration limits.
- Bypassing approval limits.
- Unexpected rail termination.
- Abusing validators escape hatch.
- Design issues.
- Take advantage of arithmetic errors.
- Private key compromise, rug-pull.

4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **Filecoin** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests using slither.
- Custom scripts (e.g. unit tests) to verify scenarios from initial threat modeling.
- **Manual review of the code.**

4.5. Disclaimer

Smart contract security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Vulnerabilities

[FIL-1132b525-L01] Reduced operator's available lockupAllowance

LOW ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team decided to postpone the fix due to issue's non-criticality, project timelines and resource constrains, but it is planned to be implemented in the next release. The vulnerable scenario has been added to the Known Issues in the documentation.

Occurences

- FilecoinPayV1.sol (#L1297-L1318, #L980-L1059)

Description

When an operator modifies the payment rate on a terminated rail before finalization, the `lockupUsage` tracking in the operator's approval becomes incorrect. The `finalizeTerminatedRail` function calculates the lockup to release using `rail.paymentRate * rail.lockupPeriod`, which assumes a constant rate for the entire lockup period.

However, when the rate is modified on a terminated rail, different rates apply to different portions of the period. The rate modification correctly uses `remainingEpochs` for adjustments, but finalization uses the full `lockupPeriod` with only the final rate, causing a mismatch that leaves phantom `lockupUsage` that is never released.

Vulnerable scenario

The following steps lead to the described result:

① Operator creates a rail with `rate=100` and `lockupPeriod=10`, resulting in `lockupUsage =1000`.
② Operator terminates the rail at block N, setting `endEpoch=N+10`. The `lockupUsage` remains 1000.
③ At block N+2 (8 epochs remaining), operator reduces rate to 50. The `lockupUsage` is correctly adjusted: `1000 - (100*8)+ (50*8)= 600`.
④ At block N+11, the rail is settled and finalized.
⑤ During finalization, `oldLockup` is calculated as `50 * 10 = 500` (using current rate for full period).

⑥ The `lockupUsage` becomes `600 - 500 = 100` instead of the expected 0.

Result: Reduced operator's available `lockupAllowance` until the payer manually increases it. The leak compounds with each rate modification on terminated rails, potentially exhausting the operator's capacity to create new rails despite having no active lockup commitments.

Recommendation

Track the actual lockup contribution per rail rather than recalculating at finalization. On each rate modification, not only update the lockup usage in approval and `lockupCurrent` in the account, but also update the rail's current lockup contribution and use that stored value at finalization instead of recalculating.

References

1. SCSVS G7: Arithmetic

[FIL-1132b525-L02] Missing validation note

LOW ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team decided to postpone the fix due to issue's non-criticality, project timelines and resource constrains, but it is planned to be implemented in the next release.

Occurences

- FilecoinPayV1.sol#L1364

Description

The `_settleWithRateChanges` function processes multiple settlement segments in a loop, where each segment can return a validation note from the `_settleSegmentGross` function. However, the implementation only preserves the validation note from the last processed segment, overwriting any previous notes.

This occurs because the `note` variable is directly assigned on each iteration (line 1364) without accumulating or concatenating the notes from all segments. When a payment rail has rate changes that require processing multiple segments, validation notes from earlier segments are lost.

Vulnerable scenario

The following steps lead to the described result:

① A payment rail has multiple rate changes, requiring the `_settleWithRateChanges` function to process multiple segments in a loop.
② Each segment calls `_settleSegmentGross`, which may return a validation note from the validator (if one is configured).
③ The function overwrites the `note` variable on each iteration, keeping only the last segment's validation note.
④ The function returns only the validation note from the final processed segment, losing all previous validation notes.

Result: Validation notes from earlier segments are lost.

Recommendation

Instead of overwriting the `note` variable, accumulate validation notes from all segments. This can be done by concatenating notes with a delimiter (e.g., `note = string.concat(note, string.concat(", ", validationNote)`, similarly as it is done in `checkAndFinalizeTerminatedRail` function (which however misses the delimiter).

References

1. SCSVS G1: Architecture, design and threat modeling
2. SCSVS G4: Business logic

[FIL-1132b525-L03] Blocking transactions using permit directly

LOW ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team decided to postpone the fix due to issue's non-criticality, project timelines and resource constrains, but it is planned to be implemented in the next release.

Occurences

- FilecoinPayV1.sol#L518

Description

The `_depositWithPermit` function calls the `permit` function directly without error handling, making it vulnerable to front-running attacks. When a user submits a `depositWithPermit` transaction, an attacker monitoring the mempool can front-run it by executing the permit themselves.

Since EIP-2612 permits can only be used once (nonces increment atomically), the attacker's transaction consumes the permit's nonce, causing the user's original `depositWithPermit` transaction to revert due to nonce mismatch. The permit call is not wrapped in try-catch, so any failure in the permit execution causes the entire deposit transaction to fail, resulting in denial of service for legitimate users attempting to deposit tokens.

Attack scenario

The attackers might take the following steps in turn:

① Monitor the mempool for pending `depositWithPermit` transactions containing permit signatures.
② Extract the permit parameters (owner, spender, amount, deadline, v, r, s) from the pending transaction.
③ Front-run the user's transaction by submitting their own transaction that calls `token.permit(to, address(this), amount, deadline, v, r, s)` directly on the token contract.
④ The attacker's transaction executes first, consuming the permit's nonce and granting approval to the FilecoinPayV1 contract.
⑤ The user's original `depositWithPermit` transaction reverts because the permit was already used (nonce mismatch or permit already executed).

Result of the attack: The user's deposit transaction fails with gas loss, and the permit is consumed without completing the deposit.

Recommendation

Wrap the permit call in a try-catch block to handle permit failures gracefully. If the permit fails, check the approval. If approval is sufficient, proceed with the deposit.

References

1. SCSVS G8: Denial of service
2. SCSVS I2: Token

[FIL-1132b525-L04] Unauthorized deposit of WETH-like tokens

LOW ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team decided to postpone the fix due to issue's non-criticality, project timelines and resource constrains, but it is planned to be implemented in the next release.

Occurences

- FilecoinPayV1.sol#L518

Description

The `depositWithPermit` function does not validate that `msg.sender` matches the permit signer (`to` address), unlike the `depositWithPermitAndApproveOperator` variants which use the `validateSignerIsRecipient` modifier. This allows anyone to call `depositWithPermit` on behalf of any user by using the user's permit signature.

When used with tokens like WETH (Wrapped ETH) or similar tokens that have empty `fallback` function, an attacker can forge a user's permit and deposit tokens into the user's account without the user's direct authorization for that specific transaction.

While this does not result in direct theft (tokens are credited to the user's account), the attacker can consume the user's approval.

Attack scenario

The attackers might take the following steps in turn:

① Call `depositWithPermit` with the forged user's permit parameters (token, to, amount, deadline, v, r, s) where `to` is the user's address.
② The permit executes without any change.
③ The `transferIn` function transfers tokens from the user's address to the contract using existing approval (and clearing it).
④ The user's account is credited with the tokens, but the user's existing approval state may be modified or cleared depending on the current approval state.

Result of the attack: Unauthorized deposit on behalf of user.

Recommendation

Add the `validateSignerIsRecipient(to)` modifier to the `depositWithPermit` function, consistent with the `depositWithPermitAndApproveOperator` variants. This will ensure that only the permit signer can execute their own permit, preventing third parties from executing permits on behalf of users.

References

1. SCSVS G5: Access control
2. SCSVS I2: Token

[FIL-1132b525-L05] Fix validation check

LOW ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team decided to postpone the fix due to issue's non-criticality, project timelines and resource constrains, but it is planned to be implemented in the next release.

Occurences

- FilecoinPayV1.sol#L1468

Description

The `_settleSegmentGross` function incorrectly validates lockup sufficiency by comparing: `payer.lockupCurrent >= grossSettledAmount` instead of comparing against `requiredLockup`.

The `grossSettledAmount` can be modified by a validator to be less than the actual `requiredLockup` that will be deducted. When a validator reduces the settlement amount (`grossSettledAmount`) but the settlement duration remains the same, `requiredLockup` (calculated as `rate * actualSettledDuration`) can exceed `grossSettledAmount`.

The validation check passes because it compares against the smaller `grossSettledAmount`, but then the larger `requiredLockup` is deducted from `lockupCurrent`, potentially causing the lockup to become insufficient or violate the lockup invariants. This creates a mismatch between what is validated and what is actually deducted, leading to potential underflow, invariant violations, or incorrect lockup state.

Result: The lockup validation passes incorrectly, and attempting to deduct more lockup than available causes a revert.

Recommendation

Calculate `requiredLockup` before the validation check and compare `payer.lockupCurrent >= requiredLockup` instead of comparing against `grossSettledAmount`.

References

1. SCSVS G4: Business logic
2. SCSVS G7: Arithmetic

[FIL-1132b525-L06] Rate queue array never cleared

LOW ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team decided to postpone the fix due to issue's non-criticality, project timelines and resource constrains, but it is planned to be implemented in the next release.

Occurences

- RateChangeQueue.sol#L27

Description

The `dequeue` function in `RateChangeQueue` contains a logic flaw. The array storage is never cleared when the queue becomes empty. After deleting an element at `queue.head`, the function checks `isEmpty(queue)` to determine if the array should be cleared. However, `isEmpty` returns `queue.head == queue.changes.length`, and at the point of checking, `queue.head` is still at the index of the just-deleted element (e.g., `length - 1` if it was the last element), while `queue.changes.length` remains unchanged (since `delete` does not reduce array length).

This means `isEmpty` always returns false immediately after deletion, even when dequeuing the last element, causing the code to increment `head` instead of clearing the array.

Result: The array storage accumulates zeroed elements that are never cleared, leading to increased gas costs and unnecessary storage usage.

Recommendation

Increment the head before checking if the queue is empty.

```
25 delete c[queue.head];
```

```
26
27 queue.head++;
28 if (isEmpty(queue)) {
29     queue.head = 0;
30     // The array is already empty, waste no time zeroing it.
31     assembly {
32         sstore(c.slot, 0)
33     }
34 }
35
36 return change;
```

References

1. SCSVS G10: Gas usage & limitations
2. SCSVS G4: Business logic

[FIL-1132b525-L07] Invalid order of parameters in errors

LOW ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team decided to postpone the fix due to issue's non-criticality, project timelines and resource constrains, but it is planned to be implemented in the next release.

Occurences

- FilecoinPayV1.sol (#L1039, #L1215, #L1463, #L1774)

Description

Four error calls in `FilecoinPayV1.sol` have incorrect parameter order compared to their error definitions in `Errors.sol`. The affected errors are:

- `LockupRateLessThanOldRate` is called with `(railId, rail.from, oldRate, payer.lockupRate)` but the error definition expects `(uint256 railId, address from, uint256 lockupRate, uint256 oldRate)`, meaning `oldRate` and `payer.lockupRate` are swapped.
- `CannotSettleFutureEpochs` is called with `(railId, untilEpoch, block.number)` but the error definition expects `(uint256 railId, uint256 maxAllowedEpoch, uint256`

`attemptedEpoch`), meaning `untilEpoch` and `block.number` are swapped.

- `InsufficientFundsForSettlement` is called with `(rail.token, rail.from, grossSettledAmount, payer.funds)` but the error definition expects `(IERC20 token, address from, uint256 available, uint256 required)`, meaning `grossSettledAmount` and `payer.funds` are swapped.
- `InsufficientNativeTokenForBurn` is called with `(msg.value, auctionPrice)` but the error definition expects `(uint256 required, uint256 sent)`, meaning `msg.value` and `auctionPrice` are swapped.

When these errors are triggered, the error messages will display incorrect parameter values, causing confusion for developers debugging issues and users trying to understand why their transactions failed.

Result: Error messages display incorrect parameter values, making it difficult to diagnose issues and potentially causing users to take incorrect corrective actions based on misleading information.

Recommendation

Fix the parameter order in all four error calls to match their error definitions exactly.

References

1. SCSVS G11: Code clarity
2. SCSVS G4: Business logic

6. Recommendations

[FIL-1132b525-R01] Unused error definitions in Errors

INFO ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team has decided to postpone implementing the recommendation due to project schedule and resource constraints, but it is planned to be implemented in the next release.

Description

Unused code increases maintenance burden while potentially confusing auditors and developers about intended functionality.

The `Errors.sol` library defines two error types that are never used anywhere in the codebase:

1. `OnlyRailParticipantAllowed`
2. `FeeWithdrawalNativeTransferFailed`

Recommendation

Remove the unused error definitions from `Errors`.

References

1. SCSVS G1: Architecture, design and threat modeling

[FIL-1132b525-R02] Use custom errors instead of revert strings

INFO ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team has decided to postpone implementing the recommendation due to project schedule and resource constraints, but it is planned to be implemented in the next release.

Description

The `RateChangeQueue` library uses string error messages in `require` statements. Three functions (`dequeue`, `peek`, and `peekTail`) use the string `"Queue is empty"` when reverting.

String error messages in `require` statements are less gas-efficient than custom errors because they are stored as bytecode, increasing both deployment costs and runtime gas consumption. Additionally, string error messages cannot carry typed parameters, limiting the information available when debugging or handling errors.

Custom errors, introduced in Solidity 0.8.4, provide a more efficient and type-safe way to handle reverts, reducing gas costs and improving code clarity.

Recommendation

Define a custom error in the `Errors` library (or locally in the `RateChangeQueue` library) for the empty queue condition, such as `error QueueEmpty()`, and replace all three `require` statements with:

- `if (queue.head >= queue.changes.length)revert Errors.QueueEmpty();`
- or `revert QueueEmpty();` if defined locally.

References

1. SCSVS G10: Gas usage & limitations
2. SCSVS G11: Code clarity

[FIL-1132b525-R03] Clear serviceFeeRecipient

INFO ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team has decided to postpone implementing the recommendation due to project schedule and resource constraints, but it is planned to be implemented in the next release.

Description

The `_zeroOutRail` function is responsible for clearing all fields of a `Rail` struct when a rail is finalized, marking it as inactive. However, the function does not clear the `serviceFeeRecipient` field, which is an address field in the `Rail` struct.

While this does not cause functional issues (since `rail.from` being `address(0)` marks the rail as inactive), it leaves the `serviceFeeRecipient` field with its previous value. This incomplete

cleanup can lead to unnecessary storage reads, potential confusion when inspecting finalized rails, and inconsistency in the zeroing-out process.

All other address fields (`token`, `from`, `to`, `operator`, `validator`) and numeric fields are properly cleared, making the omission of `serviceFeeRecipient` an oversight in the cleanup logic.

Recommendation

Add `rail.serviceFeeRecipient = address(0);` after line 1598 to clear the `serviceFeeRecipient` field along with the other rail fields.

References

1. SCSVS G11: Code clarity

[FIL-1132b525-R04] Consider using newest and clearly specified Solidity version

INFO ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team has decided to postpone implementing the recommendation due to project schedule and resource constraints, but it is planned to be implemented in the next release.

Description

In accordance with the best security practices, it is recommended to use the latest stable versions of major Solidity releases.

Very often, older versions contain bugs that have been discovered and fixed in newer versions. Moreover, it is worth remembering that the version should be **clearly specified** so that all tests and compilations are performed with the same version.

The current implementation uses:

```
pragma solidity ^0.8.27;
```

Recommendation

Use the latest stable version of major Solidity release:

```
pragma solidity 0.8.33;
```

Note: If it is planned to deploy on multiple chains, stay aware that some of them don't support `PUSH0` opcode. If solc `>=0.8.20` is used, the `PUSH0` opcode will be present in the bytecode. In this situation, it is recommended to choose `0.8.19`.

References

1. SCSVS G1: Architecture, design and threat modeling
2. Floating pragma

[FIL-1132b525-R05] Fix NatSpec comment mislabelling caller

INFO ACKNOWLEDGED

Retest (2026-02-16)

The FilOz team has decided to postpone implementing the recommendation due to project schedule and resource constraints, but it is planned to be implemented in the next release.

Description

The NatSpec documentation for `settleTerminatedRailWithoutValidation` incorrectly states that the function "may only be called by the payee". However, the actual access control is enforced by the `onlyRailClient` modifier which checks `rails[railId].from == msg.sender`, where `from` is the **payer** address.

This inconsistency could mislead developers or integrators who rely on NatSpec documentation to understand function access control.

Recommendation

Update the NatSpec comment at L1144 to correctly identify the caller as **payer**.

- FilecoinPayV1.sol#L1144

References

1. SCSVS G11: Code clarity

7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

8. Long-term best practices

8.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

8.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

8.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

8.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

8.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com