MiloTruck

Splits

Security Review

October 6, 2025

Contents

1 Introduction

1.1 About MiloTruck

MiloTruck is an independent security researcher, primarily working as a Lead Security Researcher at Spearbit and Cantina. Previously, he was part of the team at Renascence Labs and a Lead Auditor at Trust Security.

For private audits or security consulting, please reach out to him on Twitter *@milotruck*.

1.2 Disclaimer

A smart contract security review **can never prove the complete absence of vulnerabilities**. Security reviews are a time, resource and expertise bound effort to find as many vulnerabilities as possible. However, they cannot guarantee the absolute security of the protocol in any way.

2 Risk Classification

Severity Level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality.
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality/availability.
- Low - Funds are **not** at risk.

2.2 Likelihood

- High - Highly likely to occur.
- Medium - Might occur under specific conditions.
- Low - Unlikely to occur.

3 Executive Summary

3.1 About Smart Vaults

Splits is a set of composable, open-source, and audited smart contracts that make it easy to manage onchain revenue. Running exactly at gas and charging no protocol fees, it takes the form of a hyperstructure that will run for free, forever, without any maintenance or trusted third parties.

For more information, visit `https://splits.org/`.

3.2 Overview

Project Name	Smart Vaults
Project Type	Account Abstraction
Language	Solidity
Repository	0xSplits/splits-contracts-monorepo/packages/smart-vaults
Commit Hash	dbc09cd8ed34e5f7003918eeaf8466eaf26cd894

3.3 Issues Found

High	0
Medium	0
Low	2
Informational	2

4 Findings

4.1 Low Risk

4.1.1 `msg.value` isn't forwarded to fallback managers

Context: FallbackManager.sol#L76

Description: In the fallback function of `FallbackManager`, ETH isn't forwarded to the `handler` address (i.e. the call doesn't send `msg.value` below):

```
(bool success, bytes memory result) = handler.call(data);
```

As a result, fallback managers will not be able to use ETH. Calling a function for a fallback manager with value will result in ETH retained in the account.

Recommendation: Forward `msg.value` when performing the call:

```
- (bool success, bytes memory result) = handler.call(data);
+ (bool success, bytes memory result) = handler.call{value: msg.value}(data);
```

Splits: Acknowledged. This was not intended, but we will work around this limitation for future fallback managers.

MiloTruck: Acknowledged.

4.1.2 `ERC1271` cannot be used with `name` and `version` longer than 32 bytes

Context: ERC1271.sol#L12

Description: In the `ERC1271` contract, OpenZeppelin's `EIP712` (instead of `EIP712Upgradeable`) is inherited:

```
abstract contract ERC1271 is EIP712 {
```

However, `EIP712` should not be used with proxies as it stores the `name` and `version` in storage if they are longer than 32 bytes. There's no issue in the current implementation, but if `name_` or `version_` is ever upgraded to be longer in the future, `EIP712.eip712Domain()` will return an empty `name`/`version` when called.

For example:

1. Change the name of `SmartVault` to longer than 32 bytes.
2. Add the test below to `SmartVaultFactory.t.sol` and run it.

```
function test_createAccountLongName() public {
    SmartVault deployedVault = smartVaultFactory.createAccount(root, signers, 1, 0);
    (, string memory name, , , , , ) = deployedVault.eip712Domain();

    console2.log(name);
}
```

The result is that `eip712Domain()` returns an empty string instead of the actual name.

Recommendation: Consider adding a comment that the `name` and `version` should always be shorter than 32 bytes.

Splits: Acknowledged. We will ensure `name` and `version` is always shorter than 32 bytes.

MiloTruck: Acknowledged.

4.2 Informational

4.2.1 Implicit assumption that `signatures.length` is equal to the account's threshold is dangerous

Context:

- SmartVault.sol#L243-L249
- SmartVault.sol#L255-L261
- SmartVault.sol#L406-L409

Description: In `SmartVault.validateUserOp()`, there is an implicit assumption that `signatures.length == threshold`, which isn't always true since the caller can always specify more/less signatures than `threshold`. For example:

```
if (signature.signatures.length > 1) {
    isValidMerkleProof = isValidMerkleProof
        && MerkleProof.verify(signature.lightMerkleProof, signature.lightMerkleTreeRoot, lightHash_);
}
```

If `validateUserOp()` is called with `signatures.length != threshold`, logic in the `signature.signatures.length > 1` branch as shown above could run when it is not supposed to. Cases which aren't an issue:

1. If `signatures.length > threshold > 1`, extra signatures are just ignored.
2. If `signatures.length < threshold`, `MultiSigner.isValidSignature()` will eventually revert.

The interesting case to consider is `signatures.length > threshold == 1`, which means the caller specifies extra signatures while `threshold == 1`.

For the `MerkelizedUserOp` path, the caller can force `validateUserOp()` to return `INVALID_SIGNATURE` by intentionally failing the light merkle root check (e.g. set `signature.lightMerkleTreeRoot` as `bytes32(0)`). However, this isn't exploitable/abusable in practice since the `EntryPoint` reverts anyways.

Recommendation: Nevertheless, consider adding a `signatures.length == threshold` check in `MultiSigner.isValidSignature()` for safety.

Splits: Acknowledged.

MiloTruck: Acknowledgted.

4.2.2 `MultiSigner.isValidSignature()` allows duplicate signers when passing invalid signatures

Context: MultiSigner.sol#L281-L285

Description: In `MultiSigner.isValidSignature()`, `alreadySigned` is not updated when a signature is invalid (i.e. `$_.signers[signerIndex].isValidSignature()` returns `false`):

```
if ($_.signers[signerIndex].isValidSignature(frontHash_, signatures_[i].signatureData)) {
    alreadySigned |= mask;
} else {
    isValid = false;
}
```

As such, it is possible to have duplicate signers in the `signatures_` array without reverting. However, in practice this does not have any impact since `isValidSignature()` would return `false` anyways.

Recommendation: Consider immediately returning `false` instead of setting `isValid = false` in the `else` condition.

Splits: Acknowledged. We avoid returning `false` immediately for gas estimation purposes - when a dummy signature is passed, all logic should still be executed.

MiloTruck: Acknowledged.