TRUFY

Smart Contract Audit Report
for
Futures Lending

Final Report

April, 2025

Contents

1 Introduction

Trufy has been engaged by what to perform a security audit of the Trava Lending Pool smart contracts. The purpose of this audit is to achieve the followings:

- Ensure that smart contract functions work as intended.
- Identify possible vulnerabilities, which could be exploited by an attacker.
- Identify smart contract bugs, which might lead to unexpected behavior.
- Make recommendations to improve code safety and readability.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage.

1.1 About Futures Lending

1.1.1 Project Summary

- Project Name: Trave Lending Pool
- Language: Solidity
- Codebase: https://github.com/TravaLendingPool/trava-protocol-v2
- Audit method: Manual Review
- Scope:
  - `feature/admin-contract/contracts`
  - `feature/omni-lending/contracts`

1.2 Vulnerability Summary

Severity	# of Findings
Critical	1
Medium	0
Low	1
Info	2

2 Findings

ID	Title	Type	Severity	Status
ID-01	Invalid Borrow Validation Using `user` Instead of `onBehalfOf`	Logical Issue	Critical	Solved
ID-02	Index Out Of Bound In Chains Loop	Logical Issue	Low	Solved
ID-03	Misleading Variable Name `onBehalfOf` in Collateral Configuration	Logical Issue	Info	Solved
ID-04	Inconsistent Parameter Ordering Between Function and Struct	Logical Issue	Info	Solved

2.1 ID-01: Invalid Borrow Validation Using user Instead of onBehalfOf

Type	Severity	Location	Status
Logical Issue	Critical	CrossChainLendingController.sol#L475	Solved

2.1.1 Description

The function _processValidateBorrowMessage handles cross-chain borrow requests by decoding message payloads and calling validateBorrow to ensure borrowing conditions are met. However, it incorrectly uses user instead of onBehalfOf when retrieving the borrower's data:

```
(vars.user, vars.onBehalfOf, vars.asset, vars.amountToBorrow) =
    abi.decode(
    data,
    (address, address, address, uint256)
);

// ...

DataTypes.UserGlobalData storage userData = _users[vars.user];

// ...

ValidationLogic.validateBorrow(
    poolData.reserves[vars.asset],
    vars.amountToBorrow,
    vars.amountInUSD,
    _pools,
    userData, // @audit this should be userData of `onBehalfOf`
    _chainsList,
    _chainsCount,
    _addressesProvider.getPriceOracle()
);
```

This call uses userData = _users[vars.user], which is the initiator of the message, instead of _users[vars.onBehalfOf], who is the actual target borrower of the requested loan. This creates a **logical inconsistency** where the health factor, collateral, and eligibility checks are performed on the wrong account.

2.1.2 Recommendations

- Replace the borrower context in the validation call to use onBehalfOf:

```
DataTypes.UserGlobalData storage userData = _users[vars.
   onBehalfOf];
```

2.2 ID-02: Incorrect Loop Boundary When Accessing Mapping

Type	Severity	Location	Status
Logical Issue	Low	GenericLogic.sol#L209	Solved

2.2.1 Description

The function `calculateUserAccountData` iterates over a list of chains stored as a `mapping(uint256 => uint256)` using the loop condition `i <= chainsCount`:

```
for (vars.i = 0; vars.i <= chainsCount; vars.i++) {
    vars.chainId = chainsList[vars.i];
    ...
}
```

In Solidity, accessing a mapping with a key that has never been written to returns the default value `0`. This means that when `vars.i == chainsCount`, the code accesses `chainsList[chainsCount]`, which likely resolves to `0`. Unless `0` is a valid chain ID, this introduces unnecessary computation on default data.

Fortunately, all downstream computations will treat the data associated with chain ID `0` as zeroed values (due to how uninitialized structs behave in Solidity), resulting in no harmful effect on user balances or health factor.

2.2.2 Recommendations

- Change the loop condition to use a strict less-than comparison:

```
for (vars.i = 0; vars.i < chainsCount; vars.i++) { ... }
```

2.3 ID-03: Misleading Variable Name `onBehalfOf` in Collateral Configuration

Type	Severity	Location	Status
Informational	Info	CrossChainLendingController.sol#L135	Solved

2.3.1 Description

In the function `_processValidateSetUserUseReserveAsCollateral`, the input data is decoded as follows:

```
135 (address onBehalfOf, address asset, bool useAsCollateral) = abi.decode(
136     data,
137     (address, address, bool)
138 );
```

The variable `onBehalfOf` represents the target user whose collateral configuration is being updated. However, the name `onBehalfOf` implies that the action is being performed by a third party on the user's behalf. This is misleading, as there is no delegation or external actor indicated elsewhere in the function — the user in question is the direct subject of the configuration update.

All subsequent interactions refer to this address as the owner of the updated data:

```
147 DataTypes.UserGlobalData storage userData = _users[onBehalfOf];
148 ...
149 UserChainData.userConfig.setUsingAsCollateral(...);
```

2.3.2 Recommendations

- Rename the variable `onBehalfOf` to `user` to improve clarity and semantic correctness.
- Avoid naming patterns that suggest proxy or delegated behavior unless such mechanisms are implemented and enforced.

2.4 ID-04: Inconsistent Parameter Ordering Between Function and Struct

Type	Severity	Location	Status
Informational	Info	OmniLendingPool.sol#L206	Solved

2.4.1 Description

In the public function `borrow`, the parameters are defined as follows:

```
function borrow(
    address asset,
    uint256 amount,
    address onBehalfOf,
    uint16 referralCode
)
```

However, when passing these parameters into the `ExecuteBorrowParams` struct for internal processing, the ordering is reversed:

```
DataTypes.ExecuteBorrowParams({
    asset: asset,
    amount: amount,
    referralCode: referralCode,
    onBehalfOf: onBehalfOf //comes after
});
```

Although the named arguments ensure correct mapping at runtime, this inconsistency introduces cognitive overhead and potential confusion for developers, auditors, and contributors reading the codebase. It also increases the risk of mistakes if the struct is ever instantiated positionally, or in code-generated interfaces and bindings.

2.4.2 Recommendations

- Align the order of parameters in the public `borrow()` function to match the struct field order, or vice versa.

3 Appendix

3.1 Severity Definitions

Critical

This level vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.

Medium

This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to critical-risk severity.

Low

This level vulnerabilities should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.

Info

This level vulnerabilities can be ignored. They are code style violations and informational statements in the code. They may not affect the smart contract execution.

3.2 Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Centralization

Centralization findings detail the design choices of designating privileged roles or other centralizedcontrols over the code.