VERILOG

2024

UNTANGLE FINANCE: SECURITY REVIEW REPORT

Apr 8, 2024

Contents

1 | Introduction

Figure 1.1: Untangled Finance Report Cover

This report presents our engineering engagement with the Untangled Finance team on the Untangled Protocol, a digital securitization platform bridging real-world assets to decentralized finance.

Project Name	**Untangled Finance**
Repository Link	https://github.com/untangledfinance/untangled-protocol-v2
First Commit Hash	First: 292f280b;
Final Commit Hash	Final: 0662c28;
Language	Solidity
Chain	Celo

2 | About Verilog Solutions

Founded by a group of cryptography researchers and smart contract engineers in North America, Verilog Solutions elevates the security standards for Web3 ecosystems by being a full-stack Web3 security firm covering smart contract security, consensus security, and operational security for Web3 projects.

Verilog Solutions team works closely with major ecosystems and Web3 projects and applies a quality-above-quantity approach with a continuous security model. Verilog Solutions onboards the best and most innovative projects and provides the best-in-class advisory services on security needs, including on-chain and off-chain components.

3 | Service Scope

3.1 | Service Stages

Our auditing service includes the following two stages:

- Smart Contract Auditing Service

3.1.1 | Smart Contract Auditing Service

The Verilog Solutions team analyzed the entire project using a detailed-oriented approach to capture the fundamental logic and suggested improvements to the existing code. Details can be found under Findings And Improvement Suggestions.

3.2 | Methodology

- **Code Assessment**
  - □ We evaluate the overall quality of the code and comments as well as the architecture of the repository.
  - □ We help the project dev team improve the overall quality of the repository by providing suggestions on refactorization to follow the best practices of Web3 software engineering.
- **Code Logic Analysis**
  - □ We dive into the data structures and algorithms in the repository and provide suggestions to improve the data structures and algorithms for the lower time and space complexities.
  - □ We analyze the hierarchy among multiple modules and the relations among the source code files in the repository and provide suggestions to improve the code architecture with better readability, reusability, and extensibility.
- **Business Logic Analysis**
  - □ We study the technical whitepaper and other documents of the project and compare its specifications with the functionality implemented in the code for any potential mismatch between them.
  - □ We analyze the risks and potential vulnerabilities in the business logic and make suggestions to improve the robustness of the project.
- **Access Control Analysis**
  - □ We perform a comprehensive assessment of the special roles of the project, including their authorities and privileges.
  - □ We provide suggestions regarding the best practice of privilege role management according to the standard operating procedures (SOP).
- **Off-Chain Components Analysis**
  - □ We analyze the off-chain modules that are interacting with the on-chain functionalities and provide suggestions according to the SOP.
  - □ We conduct a comprehensive investigation for potential risks and hacks that may happen on the off-chain components and provide suggestions for patches.

3.3 | Audit Scope

Our auditing for Untangled Finance covered the Solidity smart contracts under the folder 'contracts' in the repository (https://github.com/untangledfinance/untangled-protocol-v2) with commit hash **292f280b**.

4 | Project Summary

The Untangled Protocol is a decentralized lending and liquidity protocol for real-world asset collaterals. Below is a graph explaining the connections and relations between contracts. Additionally, there is some relevant information regarding the most important contracts and concepts:

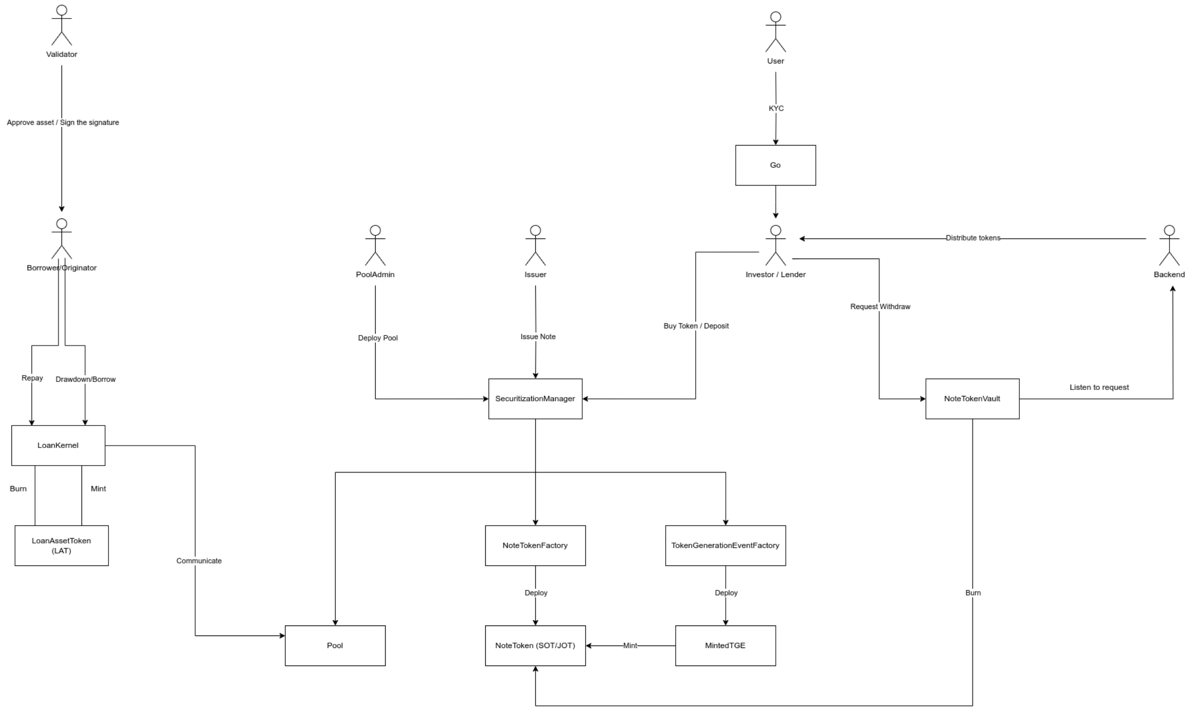

Figure 4.1: Untangled Finance Architecture

- Asset Pool

  A new instance of SecuritizationPool (Asset Pool) can be created by the approved pool creator. The pool purchases assets with its reserve in stablecoins. Asset Pool borrows from Liquidity Pool (automatically) and Pool Backers (manually) (collectively called Funders) in order to purchase Assets from Originator. The interest of Funders within an Asset Pool could be represented by a Trustee, an independent and trusted third party in the real world.

- Borrower

  An entity that borrows from the Asset Pool and uses the proceeds to develop a project, e.g. sustainability-linked loans where disbursement is made on-chain and rewards are available for achieving certain sustainability targets.

- Assets

  Each Asset purchased by Asset Pool is represented by an NFT. A yield asset (such as a green project loan) is represented by LAT. A non-yield asset (such as a trade finance invoice) is represented by AIT.

- Asset Pool Tranches

  Senior tranche financing is represented by SOT, an ERC-20 token with the currency value of 1 but is continuously compounded with interest. Junior tranche financing is represented by JOT, an ERC-20 token acting as the first loss piece in an Asset Pool.

5 | Findings and Improvement Suggestions

Severity	Total	Acknowledged	Resolved
High	3	3	3
Medium	1	1	1
Low	5	5	1
Informational	6	6	4

5.1 | High

5.1.1 | Lack of access control in `writeOff()` function

Severity	High
Source	contracts/protocol/pool/Pool.sol#L152;
Commit	292f280;
Status	Resolved in commit 67fe8b9;

- **Description**
  The `writeOff()` function is used to write off an overdue loan. As the code stands, any user could call this function to end the loan.

  In this case, some sort of access control should be implemented to avoid attackers being able to call this function.

- **Exploit Scenario**

  - □ The Untangled Finance team starts a loan;
  - □ Alice maliciously calls the `writeOff()` function to conclude the loan;
  - □ Since there is no access control on the function the loan is ended.

- **Recommendations**
  Add access control to the `writeOff()` function.

- **Results**
  Resolved in commit 67fe8b9.

  Now the `writeOff()` function is only callable by the `POOL _ADMIN _ROLE`.

5.1.2 | Incorrect balance calculation for native token

Severity	High
Source	contracts/uid/UniqueIdentity.sol#L193;
Commit	292f280;
Status	Resolved in commit 67fe8b9;

- **Description**

  The `unlockWrongToken()` function allows the admin of the contract to withdraw funds that get stuck in the contract. The balance calculation for ERC20 tokens is correct but the calculation for the native token is incorrect since the following command won't work when the input address is the zero address:

  ```
  IERC20Upgradeable(token).balanceOf(address(this))
  ```

- **Exploit Scenario**

  □ There's some native token stuck in the `UniqueIdentity` contract;

  □ The admin calls the `unlockWrongToken()` function to transfer the native token out of the contract;

  □ Since the calculation is incorrect, the native token is trapped in the contract.

- **Recommendations**

  Calculate the balance of the native token with the following command:

  ```
  address(this).balance
  ```

- **Results**

  Resolved in commit 67fe8b9.

  The suggestion was implemented.

5.1.3 | Public `_initialTGEForJOT()` function

Severity	**High**
Source	contracts/protocol/pool/SecuritizationManager.sol#L197;
Commit	292f280;
Status	Resolved in commit 67fe8b9;

- **Description**
  The `_initialTGEForJOT()` is a public function that doesn't have any access control, it just checks that the pool exists. Since this function can change the critical configuration of the JOT, it should either be internal or have some access control.
- **Exploit Scenario**
  - □ A malicious user calls the `_initialTGEForJOT()` function to start a new sale of the token;
  - □ Since the function is public the transaction goes through;
  - □ An unauthorized user was able to start the sale of a token.
- **Recommendations**
  Make the function internal.
- **Results**
  Resolved in commit 67fe8b9.

  The `_initialTGEForJOT()` function is now internal.

5.2 | Medium

5.2.1 | Lack of paused checks

Severity	**Medium**

Source	contracts/protocol/pool/SecuritizationManager.sol#L160;
	contracts/protocol/pool/SecuritizationManager.sol#L180;

Commit	292f280;
Status	Resolved in commit 67fe8b9;

- **Description**
  The `setUpTGEForSOT()` and `setUpTGEForJOT()` functions don't consider the `whenNotPaused` modifier, this means that even if the contract were to be paused, these functions could still get called. Even if these functions have some sort of access control, we recommend keeping the functions consistent with the paused state.

- **Exploit Scenario**
  State changes can be performed when the protocol is in a paused state.

- **Recommendations**
  Add the `whenNotPaused` modifier to the specified functions.

- **Results**
  Resolved in commit 67fe8b9.

  The `whenNotPaused` modifiers were added.

5.3 | Low

5.3.1 | Lack of event emission for critical operations

Severity	**Low**
Source	contracts/protocol/note-sale/MintedNormalTGE.sol#L114;
Commit	292f280;
Status	Acknowledged;

- **Description**
  Events are vital aids in monitoring contracts and detecting suspicious behavior. The `setOpeningTime()` function performs state changes, therefore they should consider the emission of an event.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Consider the emission of an event in the `setOpeningTime()` function.

- **Results**
  Acknowledged.

  Response from the Untangled team:
  "The `setOpeningTime()` doesn't play such a critical role in the entire process and also we don't need any information when calling this function so I don't think we should add an event emission here."

5.3.2 | Useless non-reentrant modifier

Severity	**Low**
Source	contracts/protocol/pool/SecuritizationManager.sol#L126;
Commit	292f280;
Status	Resolved in commit 67fe8b9;

- **Description**
  The `nonReentrant` modifier that is being used in the `_initialTGEForSOT()` function is not necessary since this function is internal and the external function that makes the call to this function already considers the `nonReentrant` modifier.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Remove the `nonReentrant` modifier of the `_initialTGEForSOT()` function.

- **Results**
  Resolved in commit 67fe8b9.

  The `nonReentrant` modifier was removed.

5.3.3 | Lack of zero address check

Severity	Low

Source	contracts/protocol/pool/SecuritizationManager.sol#L94;
	contracts/protocol/note-sale/fab/NoteTokenFactory.sol#L80;
	contracts/protocol/note-sale/fab/TokenGenerationEventFactory.sol#L106;

Commit	292f280;
Status	Acknowledged;

- **Description**
  The `_deployInstance()` function is used to deploy new contracts based on the implementation and salt provided by the admin. This function should consider checking that the returned address is different from the zero address to avoid having an incorrect configuration in the setup.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Add zero address checks whenever the `_deployInstance()` function gets used.

- **Results**
  Acknowledged.

  Response from the Untangled team:
  "We think the case where `_deployInstance()` = address(0) is nearly impossible."

5.3.4 | Repayment Router contract no longer exists

Severity	**Low**
Source	contracts/protocol/loan/LoanKernel.sol#L34-36;
Commit	292f280;
Status	Acknowledged;

- **Description**
  The `validFillingOrderAddresses` modifier performs a check on the `REPAYMENT _ROUTER` address but this contract is no longer found in the protocol.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Remove the check on the `REPAYMENT _ROUTER` contract.

- **Results**
  Acknowledged.

  Response from the Untangled team:
  "The `LoanRepaymentRouter` is merged with the `LoanKernel`, so for now, we just leave it with the same value as the LoanKernel's address."

5.3.5 | `_beforeTokenTransfer()` doesn't allow to burn tokens directly

Severity	**Low**
Source	contracts/tokens/ERC20/NoteToken.sol#L39;
Commit	292f280;
Status	Acknowledged;

- **Description**
  The `_beforeTokenTransfer()` function of the `NoteToken` contract allows minting but not burning. Although this could be changed by whitelisting the zero address, we recommend modifying the check to allow for burning directly.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Add the following check to the `require()` statement:

```
require(from == address(0) || to == address(0) ||
registryContract.isValidNoteTokenTransfer(from, to), 'Invalid transfer');
```

- **Results**
  Acknowledged.

  Response from the Untangled team:
  "Yes, we intend to do this because we don't want users to accidentally burn their NoteToken and make the protocol's calculation incorrect. Only NoteTokenVault can burn NoteToken via the redeem process."

5.4 | Informational

5.4.1 | Incorrect error message

Severity	**Informational**
Source	contracts/protocol/pool/Pool.sol#L291; contracts/protocol/pool/Pool.sol#L300;
Commit	292f280;
Status	Resolved in commit 67fe8b9;

- **Description**
  The following error message is incorrect since the `DistributionOperator` contract was removed. Instead, it should say that the check is on the `NoteTokenVault` contract.

  ```
  'SecuritizationPool: Caller must be SecuritizationManager or DistributionOperator'
  ```

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Correct the error message.

- **Results**
  Resolved in commit 67fe8b9.

  The error message was corrected.

5.4.2 | Unused `potToPool` functionality

Severity	**Informational**
Source	contracts/protocol/pool/SecuritizationManager.sol#L112-L118;
Commit	292f280;
Status	Acknowledged;

- **Description**
  The `potToPool` mapping seems no longer useful since the functions associated with it have been removed.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Remove the `potToPool` mapping and the functions related to it.
- **Results**
  Acknowledged.

  Response from the Untangled team:
  "Each pool has a corresponding pot wallet (typically EOA or MultiSig wallet) to hold the assets of that pool. The potToPool mapping is used to avoid the situation when a pot wallet is used by many pools, which could lead to fund mixing with each other."

5.4.3 | Abstract contract should be defined in a separate file

Severity	**Informational**
Source	contracts/protocol/pool/SecuritizationManager.sol#L20;
Commit	292f280;
Status	Acknowledged;

- **Description**
  The `SecuritizationManager` file in the contracts folder contains an abstract contract called `SecuritizationManagerBase` beside the actual contract. This can create issues if this file were to be imported to another contract in the future.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  We recommend keeping only the contract and creating another file for the abstract contract that can be then imported.

- **Results**
  Acknowledged.

  The Untangled Finance team decided to keep the code as it is.

5.4.4 | Unused contract

Severity	Informational
Source	contracts/protocol/factory/ProxyAdmin.sol;
Commit	292f280;
Status	Acknowledged;

- **Description**
  The `ProxyAdmin` contract imports the `ProxyAdmin` from Open Zeppelin. Since there are no changes to the Open Zeppelin implementation and this contract can be imported directly, there is no need to create an additional file.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Remove unused contracts.

- **Results**
  Acknowledged.

  The Untangled Finance team decided to keep the code as it is.

5.4.5 | TransferHelper library can be used instead of require() statement

Severity	Informational

Source	contracts/protocol/note-sale/MintedNormalTGE.sol#L174;
	contracts/protocol/note-sale/MintedNormalTGE.sol#L182;

Commit	292f280;
Status	Acknowledged;

- **Description**
  The `TransferHelper` library is imported into the logic contracts to transfer ERC20 tokens. The same could be done for other contracts that employ the same functionality.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Use the `TransferHelper` library consistently.
- **Results**
  Acknowledged.

  The Untangled Finance team decided to keep the code as it is.

5.4.6 | The `_currencyRaisedByInvestor` mapping doesn't get updated after redemption

Severity	Informational
Source	contracts/protocol/note-sale/MintedNormalTGE.sol#L124;
Commit	292f280;
Status	Acknowledged;

- **Description**
  The `_currencyRaisedByInvestor` and the `_currencyRaisedByInvestor` mappings get updated whenever an investor buys note tokens. However, when users redeem, only the `_currencyRaisedByInvestor` mapping gets updated.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Decrease the `_currencyRaisedByInvestor` mapping when users redeem.

- **Results**
  Acknowledged.

  Response from the Untangled team:
  "This struct is just to keep track of the total amount of investment for each user so there is no need to reduce its value when the user redeems the token."

6 | Use Case Scenarios

Untangled Finance provides the infrastructure to host blockchain-based credit pools, where investors – in this case, certified investors, firms, and decentralized autonomous organizations (DAOs) – deposit funds to lend and earn a yield. Depositors receive an ERC-20 token that represents their positions. The platform also features a built-in liquidation engine, a forward-looking credit assessment model, and an auction-based withdrawal mechanism for investors who want to exit the pools early.

7 | Access Control Analysis

There are different privileged roles in the Untangled protocol. A description of the roles in each of the contracts can be found below:

7.1 | SecuritizationPool

- `owner` can collect assets, set up risk scores, debt ceiling, and set min first loss cushion.
- `tgeAddress` can set up opening block timestamp.

7.2 | SecuritizationManager

- `POOL_ADMIN_ROLE` can initiate new pools.
- `owner` can initiate TGE for SOT and JOT.
- `DEFAULT_ADMIN _ROLE` can pause and unpause pools.
- `OWNER_ROLE` can update TGE info.

7.3 | NoteTokenVault

- `BACKEND_ADMIN_ROLE` can distribute the tokens between the users that have initiated redemption.

8 | Appendix

8.1 | Appendix I: Severity Categories

Severity	Description
High	Issues that are highly exploitable security vulnerabilities. It may cause direct loss of funds / permanent freezing of funds. All high severity issues should be resolved.
Medium	Issues that are only exploitable under some conditions or with some privileged access to the system. Users' yields/rewards/information is at risk. All medium severity issues should be resolved unless there is a clear reason not to.
Low	Issues that are low risk. Not fixing those issues will not result in the failure of the system. A fix on low severity issues is recommended but subject to the clients' decisions.
Information	Issues that pose no risk to the system and are related to the security best practices. Not fixing those issues will not result in the failure of the system. A fix on informational issues or adoption of those security best practices-related suggestions is recommended but subject to clients' decision.

8.2 | Appendix II: Status Categories

Severity	Description
Unresolved	The issue is not acknowledged and not resolved.
Partially Resolved	The issue has been partially resolved
Acknowledged	The Finding / Suggestion is acknowledged but not fixed / not implemented.
Resolved	The issue has been sufficiently resolved

9 | Disclaimer

Verilog Solutions receives compensation from one or more clients for performing the smart contract and auditing analysis contained in these reports. The report created is solely for Clients and published with their consent. As such, the scope of our audit is limited to a review of code, and only the code we note as being within the scope of our audit is detailed in this report. It is important to note that the Solidity code itself presents unique and unquantifiable risks since the Solidity language itself remains under current development and is subject to unknown risks and flaws. Our sole goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies. Thus, Verilog Solutions in no way claims any guarantee of security or functionality of the technology we agree to analyze.

In addition, Verilog Solutions reports do not provide any indication of the technology's proprietors, business, business model, or legal compliance. As such, reports do not provide investment advice and should not be used to make decisions about investment or involvement with any particular project. Verilog Solutions has the right to distribute the Report through other means, including via Verilog Solutions publications and other distributions. Verilog Solutions makes the reports available to parties other than the Clients (i.e., "third parties") – on its website in hopes that it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.