BLOCKSEC

Security Audit

Report for Unleash-audit-smart-contract

Date: September 13, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Unleash
Target	Unleash-audit-smart-contract

Version History

Version	Date	Description
1.0	September 13, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-09-23 08:06:29+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository of Unleash-audit-smart-contract[1] of Unleash. Note that, we did **NOT** audit all the modules in the repository. The modules this audit report covers include misc, abstracts and internal-looping-hook folder contract. Specifically, the file covered in this audit includes:

```
1 AaveOracle.sol
2 FallbackOracle.sol
3 Execution.sol
4 FlashLoanLoopingHookBase.sol
5 TokenPulling.sol
6 InternalLoopingHook.sol
```

Listing 1.1: Audit Scope for this Report

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Repository Branch	Version	Commit Hash
Unleash	core	Version 1	ba3ac29a68687257591066e430884f6a57c2bd49
		Version 2	N/A
	loopinghookandflashrepay	Version 1	3bf3ed8d15a2dc116f11373830773c2ca020b19a
		Version 2	4064ae457da93239bcaf21057c9555eddf5bde21

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any war-

[1] https://github.com/unleash-protocol/unleash-audit-smart-contract

ranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist

- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **one** potential security issue. Besides, we have **two** recommendations and **two** notes.

- Medium Risk: 1
- Recommendation: 2
- Note: 2

ID	Severity	Description	Category	Status
1	Medium	Protocol can be drained while creating an empty market	DeFi Security	Confirmed
2	-	Redundant code	Recommendation	Fixed
3	-	Simplify logic in the function `_permitDebtToken()`	Recommendation	Confirmed
4	-	Pontential centralization risk	Note	-
5	-	Pontential risk in stable interestRate mode	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Protocol can be drained while creating an empty market

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description Due to the use of `rayDiv` in the supply/withdraw process of the Unleash protocol, there is no guarantee that the rounding direction always favors the protocol. Attackers can manipulate the `LiquidityIndex` of `Atoken` in an empty market state, exploiting rounding errors to drain assets from the protocol.

```
76   /**
77   * @notice Divides two ray, rounding half up to the nearest ray
78   * @dev assembly optimized for improved gas savings, see https://twitter.com/transmissions11/
         status/1451131036377571328
79   * @param a Ray
80   * @param b Ray
81   * @return c = a raydiv b
82   */
83  function rayDiv(uint256 a, uint256 b) internal pure returns (uint256 c) {
84    // to avoid overflow, a <= (type(uint256).max - halfB) / RAY
85    assembly {
86      if or(iszero(b), iszero(iszero(gt(a, div(sub(not(0), div(b, 2)), RAY))))) {
87        revert(0, 0)
88      }
89
```

```
90      c := div(add(mul(a, RAY), div(b, 2)), b)
91    }
92  }
```

Listing 2.1: WadRayMath.sol

Impact The funds in the protocol will be drained.

Suggestion Supply a certain amount of assets in the same transaction of creating a new market.

Feedback from the Project The team will adjust the deployment script to resolve this issue.

2.2 Additional Recommendation

2.2.1 Redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `InternalLoopingHook` contract inherits from the `Ownable2Step` contract in line 19, but none of the functions implement the access control, the `Ownable2Step` contract is redundant.

The contract `InternalLoopingHook` contains a `receive()` function, but according to the protocol design, the contract will not receive native tokens through this function. Additionally, `flashloan` do not support transferring native tokens. The `receive()` function is redundant.

```
17  contract InternalLoopingHook is
18  IInternalLoopingHook,
19  Ownable2Step,
20  Execution,
21  TokenPulling,
22  FlashLoanLoopingHookBase
```

Listing 2.2: InternalLoopingHook.sol

```
129  receive() external payable {
130      // solhint-disable-previous-line no-empty-blocks
131  }
```

Listing 2.3: InternalLoopingHook.sol

Suggestion Remove the redundant code.

2.2.2 Simplify logic in the function `_permitDebtToken()`

Status Confirmed

Introduced by `Version 1`

Description In the function `_permitDebtToken()`, the corresponding debtToken is retrieved based on the `interestRateMode` parameter provided by the user in line 63. However, according

to the protocol design and Aave's related security risk warnings[1], `interestRateMode` should always be set to `InterestRateMode.VARIABLE`. Therefore, the logic in this function is redundant.

```
55  function _permitDebtToken(
56      address delegator,
57      address token,
58      uint256 interestRateMode,
59      TokenApproveSig memory sig
60  ) internal {
61      DataTypes.ReserveData memory reserveData = POOL.getReserveData(token);
62      address debtToken;
63      if (interestRateMode == uint8(DataTypes.InterestRateMode.STABLE)) {
64          debtToken = reserveData.stableDebtTokenAddress;
65      } else if (
66          interestRateMode == uint8(DataTypes.InterestRateMode.VARIABLE)
67      ) {
68          debtToken = reserveData.variableDebtTokenAddress;
69      }
70
71
72      ICreditDelegationToken(debtToken).delegationWithSig(
73          delegator,
74          address(this),
75          sig.approveAmount,
76          sig.expiration,
77          sig.v,
78          sig.r,
79          sig.s
80      );
81  }
```

Listing 2.4: FlashLoanLoopingHookBase.sol

Suggestion Revise the function `_permitDebtToken()` logic accordingly.

Feedback from the Project The team keeps the current approach for generalization, as disabling the stable rate has no impact on the protocol.

2.3 Note

2.3.1 Pontential centralization risk

Description There are some centralization risks in this protocol. For example, the Unleash uses upgradeable contracts, which means that if the team's private keys are leaked, malicious users can upgrade the contracts to steal user assets. Additionally, the `PoolAdmin` role can directly transfer assets from the pool and change various parameters in the market, such as LTV, price oracle, and liquidation thresholds, posing risks to user assets.

Feedback from the Project The team promises to use a multisig wallet for holding a critical role in the protocol.

[1] https://twitter.com/aave/status/1723736103012704400

2.3.2 Pontential risk in stable interestRate mode

Description Aave disclosed[2] a vulnerability related to its stable interest rate model and closed the fixed rate mode through a code patch last year. It is recommended to update the corresponding code patch to prevent potential security risks.

Feedback from the Project The team promises to disable stable interest rate mode.

[2] https://twitter.com/aave/status/1723736103012704400


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS