SECURITY AUDIT OF

BLASTOFF SMART CONTRACTS

Public Report

Mar 19, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.
Blast	Blast Blockchain is a Fast, Secure, Scalable, Low-cost, Layer two (L2) and EVM compatible Blockchain built on the Ethereum network. It is built on the Optimism rollup, inheriting the powerful OP Stack. Blast stands out from other L2 network due to its native yield feature.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Mar 19, 2024. We would like to thank the BlastOff for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the BlastOff smart contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team found some vulnerabilities in the given version of BlastOff smart contracts. BlastOff team has resolved and fixed all of these issues following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About BlastOff smart contracts

BlastOff is a revolutionary Idle Yield - launchpad and yield aggregator built on Blast.

It offers a unique combination of features that make it an ideal platform for the Blast network. Via a combination of Blast Native Yield and YZone, they generate maximum yield for their users, YIDOs act as a layer on top to generate even more returns via idle participation in funding new projects on Blast.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of BlastOff smart contracts.

It was conducted on the following commits from two git repositories:

- *https://github.com/BlastOffOrg/aggregator-vaults/commit/9a258338c4c8d2cf96dae7755922258812df1923*
- *https://github.com/BlastOffOrg/native-yield-strategy/commit/9fdb2da456a623c757dcd54edd7e073f5f84808b*

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
db88c3a3b0886d0269964655b8d40af5ceedf9b9b0b9e038f3d6dfa1203b125e	./aggregator-vaults/contracts/VaultFactory.vy
57ef13d293c715709a9f3645cf3e8dc976ef475317eacb22089a595771854b88	./aggregator-vaults/contracts/VaultV3.vy
1ac6bb25a65c90b89c60129f592914fa5bfb4c44d7f871f34e4bf030f02fdf7b	./native-yield-strategy/src/NativeYieldStrategy.sol
732dbe02a5c6e53008e60b2b106cdadbfbe78e4b2e29c264849f15d552c01150	./native-yield-strategy/src/TokenizedStrategy.sol
9a689f3f076135ec8e87fc23b0d8ccad65cbb8edc2adc8bb876e4d2643c452b8	./native-yield-strategy/src/BaseStrategy.sol

The latest version of the code was reviewed on the following commits from the `aggregator-vaults` and `native-yield-strategy` repositories:

- `aggregator-vaults`: `0b72520a89c592d1a54180dbf087cafdc7131566`
- `native-yield-strategy`: `44d4f254bf21c87366ff7eb998450f4568e9fa51`

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

BlastOff acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. BlastOff understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, BlastOff agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the BlastOff will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the BlastOff, the final report will be considered fully accepted by the BlastOff without the signature.

2. AUDIT RESULT

2.1. Overview

The BlastOff smart contracts was written in `Solidity` language. The source code was mainly forked from Yearn Finance with some minor modifications.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of BlastOff smart contracts.

BlastOff team fixed these issues, according to Verichains's draft report, by updating the source code to incorporate important updates from the original repositories of Yearn Finance version `v3.0.2`.

#	Issue	Severity	Status
1	Incorrect update of `shares_to_burn` in the `_process_report` function	HIGH	Fixed
2	Redundant assert in the `issue_shares_for_amount` function	LOW	Fixed

2.2.1. Incorrect update of `shares_to_burn` in the `_process_report` function HIGH

Affects files:

- aggregator-vaults/contracts/VaultV3.vy

The statement `shares_to_burn = 0` seems incorrect and should be replaced with `shares_to_lock = 0` as seen in the `yearn-vaults-v3` repository version `v3.0.2`.

```
@internal
def _process_report(strategy: address) -> (uint256, uint256):
    ...
    # If we will end with more shares than we have now.
    if ending_supply > total_supply:
        # Issue the difference.
        self._issue_shares(unsafe_sub(ending_supply, total_supply), self)

    # Else we need to burn shares.
    elif total_supply > ending_supply:
        # Can't burn more than the vault owns.
        to_burn: uint256 = min(unsafe_sub(total_supply, ending_supply),
total_locked_shares)
        self._burn_shares(to_burn, self)

    # Adjust the amount to lock for this period.
    if shares_to_lock > shares_to_burn:
        # Don't lock fees or losses.
        shares_to_lock = unsafe_sub(shares_to_lock, shares_to_burn)
    else:
        shares_to_burn = 0          # AUDIT: INCORRECT UPDATE
    ...
```

UPDATES

- **Mar 19, 2024**: This issue has been acknowledged and fixed by BlastOff team.

2.2.2. Redundant assert in the `issue_shares_for_amount` function LOW

Affects files:

- aggregator-vaults/contracts/VaultV3.vy

The assert statement `assert total_assets > amount, "amount too high"` is redundant and should be removed as seen in the `yearn-vaults-v3` repository version `v3.0.2`.

```
@internal
def _issue_shares_for_amount(amount: uint256, recipient: address) -> uint256:
    """
    Issues shares that are worth 'amount' in the underlying token (asset).
    WARNING: this takes into account that any new assets have been summed
    to total_assets (otherwise pps will go down).
    """
    total_supply: uint256 = self._total_supply()
    total_assets: uint256 = self._total_assets()
    new_shares: uint256 = 0

    # If no supply PPS = 1.
    if total_supply == 0:
        new_shares = amount
    elif total_assets > amount:
        new_shares = amount * total_supply / (total_assets - amount)
    else:
        # If total_supply > 0 but amount = totalAssets we want to revert because
        # after first deposit, getting here would mean that the rest of the shares
        # would be diluted to a price_per_share of 0. Issuing shares would then mean
        # either the new depositor or the previous depositors will loose money.
        assert total_assets > amount, "amount too high"

    # We don't make the function revert
    if new_shares == 0:
       return 0

    self._issue_shares(new_shares, recipient)

    return new_shares
```

UPDATES

- **Mar 19, 2024**: This issue has been acknowledged and fixed by BlastOff team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 19, 2024*	Public Report	Verichains Lab

Table 2. Report versions history