Blast Futures
Audit Report

Thu Apr 25 2024

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

Blast Futures Audit Report

1 Executive Summary

1.1 Project Information

Description	Exchange contract
Type	Staking
Auditors	ScaleBit
Timeline	Wed Apr 17 2024 - Tue Apr 23 2024
Languages	Solidity
Platform	Blast
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/BlastFutures/Blast-Futures-Exchange
Commits	013e6254e002a8c71f5a73b0278297ca4a8387f5

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
BVA	foundry/src/BfxVault.sol	69c7ec812ab35ba723e0265d12b0feb15e26bf7d
BFX	foundry/src/Bfx.sol	12d2b4d6a1a533228381c5cf11d2aa7086f03bc9
EIP7V	foundry/src/EIP712Verifier.sol	b4520ae439952dc17c6a5aa8f7450e9570d851e1
IVA	foundry/src/IVault.sol	3dc07818d6820da597e377bb3fa47dc3e96f8fe8
BDE	foundry/src/BfxDeposit.sol	2dca1f97f55d78067b238d3825335a2c9425e52d
IPD	foundry/src/IPoolDeposit.sol	2e6f93caeb4062dd85fb01d40e773697936714f9

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	0	6
Informational	3	0	3
Minor	3	0	3
Medium	0	0	0
Major	0	0	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Blast Futures to identify any potential issues and vulnerabilities in the source code of the Blast Futures smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 6 issues of varying severity, listed below.

ID	Title	Severity	Status
BDE-1	Different Licenses	Informational	Acknowledged
BDE-2	Token Setting Risk	Informational	Acknowledged
BFX-1	Withdrawal Signatures Never Expire	Informational	Acknowledged
BVA-1	Centralization Risk	Minor	Acknowledged
BVA-2	Risks of Off-chain Withdrawals	Minor	Acknowledged
BVA-3	Role Logic Conflict Minor	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the Blast Futures Smart Contract :
Foundry/src/BfxVault.sol: **Owner**

- Owner can reset `ADMIN_ROLE` through `makeOwnerAdmin` function.
- Owner can set the `onlyAdmin` function switch through the `setOwnerIsSoleAdmin` function.
- Owner can set the `claimer` address through the `setClaimer` function.
- Owner can set the `paymentToken` address through the `setPaymentToken` function.
- Owner can set the `bfx` address through the `setBfx` function.
- Owner can withdraw contract funds through the `withdrawTokensTo` function.
- Owner has `ADMIN_ROLE`.

Admin

- Admin can add any permissions to any address through the `addRole` function, such as `ADMIN_ROLE`, `TRADER_ROLE` and `TREASURER_ROLE`.
- Admin can remove any permissions to any address through the `removeRole` function, such as `ADMIN_ROLE`, `TRADER_ROLE` and `TREASURER_ROLE`.
- Admin can quickly operate permissions through the `addTrader/removeTrader` function and `addTreasurer/removeTreasurer` function.

Claimer

- Claimer can obtain `BLAST` funds to the contract through the `claimGas` function.

USER

- USER can stake funds to `bfx` through the `stake` function, and the current contract records events.

Foundry/src/BfxDeposit.sol: **Owner**

- Owner can set the `claimer` address through the `setClaimer` function.
- Owner can set the `paymentToken` address through the `setPaymentToken` function.
- Owner can set the `rabbit` address through the `setRabbit` function.
- Owner can withdraw contract funds through the `withdrawTokensTo` function.

Claimer

- Claimer can obtain `BLAST` funds to the contract through the `claimGas` function.

USER

- USER can stake funds to `rabbit` through the `individualDeposit` function, and the current contract records events.
- USER can stake funds to `rabbit` in batches through the `pooledDeposit` function, and the current contract records events.

Foundry/src/Bfx.sol: **Owner**

- Owner can set the `paymentToken` address through the `setPaymentToken` function.
- Owner can set the `external_signer` address through the `changeSigner` function.
- Owner can set the `claimer` address through the `changeClaimer` function.
- Owner can withdraw contract funds through the `withdrawTokensTo` function.

Claimer

- Claimer can obtain the `paymentToken` reward to the contract through the `claimYield` function.
- Claimer can obtain `BLAST` funds to the contract through the `claimGas` function.

USER

- USER can stake funds to the current contract through the `deposit` function.
- USER can use the signature to withdraw funds to the specified `trader` address through the `withdraw` function.

4 Findings

BDE-1 Different Licenses

Severity: Informational

Status: Acknowledged

Code Location:

foundry/src/BfxDeposit.sol#1

Descriptions:

The BFX project is expected to be released under the MIT certificate, a different certificate was selected in the file.

```
// SPDX-License-Identifier: BUSL-1.1
```

Suggestion:

It is recommended to publish it under the MIT License.

BDE-2 Token Setting Risk

Severity: Informational

Status: Acknowledged

Code Location:

foundry/src/BfxDeposit.sol#71;

foundry/src/BfxVault.sol#286;

foundry/src/Bfx.sol#149

Descriptions:

1. `paymentToken` is implemented through the call method. It does not support tokens that have no return value and do not implement standard transfers correctly. The contract does not support deflation tokens, which will trigger events with wrong balances.
2. The Owner can change the `paymentToken` address at will, which may cause some business risks.

```
function tokenCall(bytes memory data) private returns (bool) {
    (bool success, bytes memory returndata) = address(paymentToken).call(data);
    if (success) {
        if (returndata.length > 0) {
            success = abi.decode(returndata, (bool));
        } else {
            success = address(paymentToken).code.length > 0;
        }
    }
    return success;
}
```

Suggestion:

It is recommended not to set up deflationary tokens or unsupported tokens.

BFX-1 Withdrawal Signatures Never Expire

Severity: Informational

Status: Acknowledged

Code Location:

foundry/src/Bfx.sol#128

Descriptions:

The signature verification function lacks verification of expiration time.

```
bytes32 digest = _hashTypedDataV4(
    keccak256(
        abi.encode(
            keccak256(
                "withdrawal(uint256 id,address trader,uint256 amount)"
            ),
            id,
            trader,
            amount
        )
    )
);
```

Suggestion:

It is recommended to add a deadline.

BVA-1 Centralization Risk

Severity: Minor

Status: Acknowledged

Code Location:

foundry/src/BfxVault.sol#300;

foundry/src/BfxDeposit.sol#137

Descriptions:

1. The Owner can arbitrarily set the addresses of `bfx` and `rabbit`. At the same time, the Owner can withdraw the contract funds through withdrawTokensTo. The Owner can transfer the user's deposit funds by modifying the address, which has certain centralization risks.

```
function individualDeposit(address contributor, uint256 amount) external {
...
    bool success = makeTransferFrom(msg.sender, rabbit, amount);
...
function setRabbit(address _rabbit) external onlyOwner {
    rabbit = _rabbit;
...
function withdrawTokensTo(uint256 amount, address to) external onlyOwner
```

2. Owner can set the paymentToken address arbitrarily.

```
    function setPaymentToken(address _paymentToken) external onlyOwner {
        paymentToken = IERC20(_paymentToken);
        emit SetToken(_paymentToken);
    }
```

Suggestion:

It is recommended to use multi-signature addresses to alleviate this problem.

BVA-2 Risks of Off-chain Withdrawals

Severity: Minor

Status: Acknowledged

Code Location:

foundry/src/BfxVault.sol;

foundry/src/BfxDeposit.sol

Descriptions:

The contract uses the emit event to record user deposit funds. The `stake` function does not determine whether the caller is an EOA account, which may cause the off-chain signature withdrawal function to transfer funds to a contract that does not receive funds. This is recorded in the `individualDeposit` and `pooledDeposit` functions. `depositId` and `poolId` are not recorded by the sender. In `emit Deposit`, `contribAmount` is any input of the sender (greater than MIN_DEPOSIT). When there is a deposit data conflict, there is only `depositId`.

```
function stake(uint256 amount) external {
emit Stake(stakeId, msg.sender, amount);
...
function individualDeposit(address contributor, uint256 amount) external {
emit Deposit(depositId, contributor, amount, 0);
...
function pooledDeposit(Contribution[] calldata contributions) external{
emit Deposit(depositId, contribution.contributor, contribAmount, poolId);
...
emit PooledDeposit(poolId, totalAmount);
```

Suggestion:

It is recommended to mitigation measures.

BVA-3 Role Logic Conflict Minor

Severity: Minor

Status: Acknowledged

Code Location:

foundry/src/BfxVault.sol#143

Descriptions:

A USER with `ADMIN_ROLE` permissions can delete the other party's `ADMIN_ROLE` permissions. Although the owner can call the makeOwnerAdmin function to regain the `ADMIN_ROLE` permissions, when a malicious administrator deletes it, even if the owner can use the `setOwnerIsSoleAdmin` function to mitigate it, this problem still cannot be solved. Removing permissions is still risky.

```
function removeAdmin(address user) external {
    removeRole(user, ADMIN_ROLE);
}

function removeRole(address signer, uint256 role) public onlyAdmin {
    signers[signer][role] = false;
    emit RemoveRole(signer, role, msg.sender);
}

require(signers[msg.sender][ADMIN_ROLE], "NOT_AN_ADMIN");

function makeOwnerAdmin() external onlyOwner {
        signers[owner][ADMIN_ROLE] = true;
    }

function setOwnerIsSoleAdmin(bool value) external onlyOwner {
        ownerIsSoleAdmin = value;
}
```

Suggestion:

It is recommended that only the owner can remove administrator permissions, because `remove` is more risky than `add` .

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.