OptionBlitz

Smart Contract Security Audit

Prepared by ShellBoxes

Feb 2nd, 2023 - Feb 4th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	OptionBlitz
Version	1.0
Classification	Public

Scope

Contract	Address
BlxToken	0x0502F0fd4Be7854b5749328f7e3DD013B94e858E

Re-Audit

Contract	Address
BlxToken	0x0502F0fd4Be7854b5749328f7e3DD013B94e858E

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

OptionBlitz engaged ShellBoxes to conduct a security assessment on the OptionBlitz beginning on Feb 2nd, 2023 and ending Feb 4th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About OptionBlitz

OptionBlitz is a decentralised trading platform built on the blockchain. You can buy different types of options including binary options, barrier options, American options, European Options and Turbos. OptionBlitz also supports liquidity staking where clients can invest funds in exchange for a revenue share which is generated from fees collected from traders.

Issuer	OptionBlitz
Website	`https://optionblitz.co`
Type	Solidity Smart Contract
Whitepaper	`https://optionblitz.co/static/media/optionblitz_whitepaper.ce295a0d387dd3ce94b9.pdf`
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and

implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the OptionBlitz implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, this token contract is well-designed and constructed, but the implementation might be improved by addressing the discovered flaws, which include , 4 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Potential Imbalance in Token Distribution	LOW	Acknowledged
SHB.2. Approve Race Condition	LOW	Acknowledged
SHB.3. Usage of block.timestamp	LOW	Acknowledged
SHB.4. Floating pragma	LOW	Fixed

3 Finding Details

SHB.1 Potential Imbalance in Token Distribution

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

The contract deployment process assigns the entire token supply to a single address, which could result in an imbalance in token distribution. There is a risk that the designated address may be incorrect, leading to a situation where the entire supply of tokens is inaccessible.

Files Affected:

SHB.1.1: BlxToken.sol

```
1519 constructor(address mintTo) ERC20(name_, symbol_) ERC20Permit(name_) {
1520     if (mintTo == address(0)) mintTo = _msgSender();
1521     _mint(mintTo, totalSupply_);
1522 }
```

Recommendation:

It is recommended to use a multisig as the deployer of the contract to include multiple parties in the supply allocation.

Updates

The OptionBlitz team has acknowledged the issue. They have confirmed that the contract has already been deployed on the mainnet and the initial minter address is correct. Furthermore, the team has stated that the tokens will be transferred to the token sale contracts

when they are ready, and any remaining balance will be transferred to the DAO Treasury as documented in the OptionBlitz whitepaper - DECENTRALISED OPTIONS TRADING PROTOCOL.

SHB.2 Approve Race Condition

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

The standard ERC20 implementation contains a well-known racing condition in its approve function, wherein a spender can observe the token owner broadcast a transaction altering their approval ,and then quickly sign and broadcast a transaction using transferFrom to claim the current approved amount to the spender's balance. If the spender's transaction is verified prior to the owner's, the spender will be able to receive both approval amounts.

Files Affected:

SHB.2.1: BlxToken.sol

```
287 function approve(address spender, uint256 amount) public virtual
        ↪ override returns (bool) {
288     address owner = _msgSender();
289     _approve(owner, spender, amount);
290     return true;
291 }
```

Recommendation:

We recommend using increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it. This can be implemented by overriding the approve function and disabling it using a revert.

Updates

The OptionBlitz team has acknowledged the potential race condition present in the standard ERC20 implementation's approve function and decided to retain the function for compatibility reasons. The team is aware that this function is widely expected to be present in ERC20 contracts and that disabling it could result in compatibility issues with other DApps and contracts. The approve race condition is widely known in the industry, and the team believes that it is the responsibility of the users to be aware of the associated risks and choose alternative methods if they so choose.

SHB.3 Usage of block.timestamp

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 1

Description:

block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Files Affected:

SHB.3.1: BlxToken.sol

```
1444 function permit(
1445     address owner,
1446     address spender,
1447     uint256 value,
1448     uint256 deadline,
1449     uint8 v,
```

```
1450      bytes32 r,
1451      bytes32 s
1452  ) public virtual override {
1453      require(block.timestamp <= deadline, "ERC20Permit: expired deadline
              ↪ ");
```

Recommendation:

Verify that a delay of 900 seconds will not harm the logic of the contract.

Updates

The OptionBlitz team has taken the issue of using block.timestamp in their contract into consideration. They have acknowledged this issue, stating that the permit call is only intended to have a short duration, and therefore any potential inaccuracies in block.timestamp would not pose a significant risk to the overall business logic.

SHB.4 Floating pragma

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 1

Description:

The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Files Affected:

SHB.4.1: BlxToken.sol

```
1509 pragma solidity ^0.8.0;
```

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Updates

The OptionBlitz team resolved the issue by fixing the pragma version to 0.8.16 in the hardhat.config.js file.

SHB.4.2: hardhat.config.js

```
{
    version: "0.8.16",
    settings: {
// viaIR: false,
      optimizer: {
        enabled: true,
        runs: 1000000,
        details: {
          yul: true,
          yulDetails: {
           stackAllocation: true,
// optimizerSteps: "dhfoDgvulfnTUtnIf"
          }
        }
      },
    },
  }
```

4 Best Practices

BP.1 Remove unnecessary function

Description:

The verifyingContract function returns the contract's address; however, this function is redundant because the caller already needs the contract's address to invoke it. Therefore, the user will not find this function useful and it is recommended to remove it.

Files Affected:

BP.1.1: BlxToken.sol

```
1537 function verifyingContract() public view returns(address) { return
       ↪ address(this); }
```

Status - Acknowledged

The OptionBlitz team acknowledged the best-practice, stating that verifyingContract function is implemented for clarity and to inform the users that the verifying contract is the same as the token contract.

BP.2 Public functions can be declared external

Description:

The functions with a public scope that are not called inside the contract should be declared external to accurately express the visibility of the function.

Files Affected:

BP.2.1: BlxToken.sol

```
1530 function burn(uint256 amount)
1531     public override
```

```
{
    _burn(_msgSender(), amount);
}
```

BP.2.2: BlxToken.sol

```
function verifyingContract() public view returns(address) { return
    ↪ address(this); }
```

Status - Acknowledged

The OptionBlitz team acknowledged the best-practice, stating that these functions do not contain memory/calldata parameters, which implies that changing the visibility will not impact the gas cost.

BP.3 Eliminate Unnecessary Code

Description:

The IBlxToken interface includes commented code that serves no functional or documentation purposes. To maintain code clarity and efficiency, it is recommended to eliminate this unused code.

Files Affected:

BP.3.1: BlxToken.sol

```
interface IBlxToken is IERC20 {
// function update_blxusd(uint256 _blxusd) external returns (uint256);
//
// function get_blxusd() external view returns (uint256);
    function burn(uint256 amount) external;
// function burnFor(address holder, uint256 amount) external;
}
```

Status - Acknowledged

The OptionBlitz team acknowledged the best-practice, stating that removing comments from this contract will not impact the generated byte-code. The Solidity compiler only includes the necessary instructions for the execution of the contract and does not include any comments in the generated byte-code.

5 Conclusion

In this audit, we examined the design and implementation of OptionBlitz contract and discovered several issues of varying severity. OptionBlitz team addressed 1 issue raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised OptionBlitz Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

6 Scope Files

6.1 Audit

Files	MD5 Hash
BlxToken.sol	7f97e2ea8e770adf1cdae92ca8f2cd54

6.2 Re-Audit

Files	MD5 Hash
BlxToken.sol	7f97e2ea8e770adf1cdae92ca8f2cd54

7 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com