HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Blizzard
Date: October 1st, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Blizzard.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Pools
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Zip archive	blizz-contracts-audit.zip
Technical Documentation	NO
JS tests	NO
Timeline	27 SEPTEMBER 2021 - 01 OCTOBER 2021
Changelog	01 OCTOBER 2021 - INITIAL AUDIT

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Blizzard (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between September 27th, 2021 - October 1st, 2021.

Scope

The scope of the project is smart contracts in the repository:

Zip archive:
blizz-contracts-audit.zip
md5 Hash:
9fb45b6edfd2adff4ebc572bff781b28
Technical Documentation: No
JS tests: No
Contracts:
Blizzard.sol
BlizzLockedStrategy.sol
BlizzLockedVault.sol
BlizzStakePool.sol
JoeLPStrategy.sol
JOEVault.sol
LPVault.sol
PangolinLPStrategy.sol

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

According to the assessment, the Customer's smart contracts are secured.

Insecure	Poor secured	Secured	Well-secured

You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

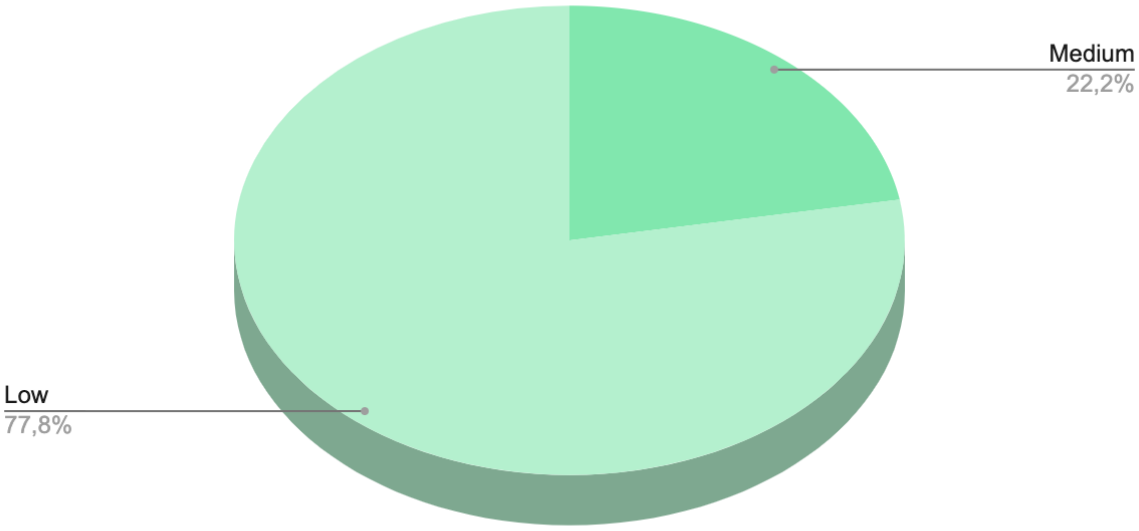
As a result of the audit, security engineers found **2** medium and **7** low severity issues.

Notice:

BlizzLockedStrategy, JoeLPStrategy, PangolinLPStrategy contracts implement the **StratManager** and **FeeManager** which are outside the scope of the current audit, so we cannot guarantee its functions and secureness.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Graph 1. The distribution of vulnerabilities after the audit.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

No critical issues were found.

High

No high severity issues were found.

Medium

1. Parts of contracts logic was not provided for audit

   Some parts of contracts logic, like @openzeppelin contracts version, interfaces declaration, **StratManager** and **FeeManager** contracts were not provided for audit. Without those parts automatic checking is very limited as well as security verification could not be done because of the hidden logic.

   Recommendation: Please consider providing the entire scope for audit.

2. Dangerous usage of **tx.origin**

   tx.origin-based protection can be abused by a malicious contract if a legitimate user interacts with the malicious contract.

Exploit Example:

```
contract TxOrigin {
   address owner = msg.sender;

   function bug() {
       require(tx.origin == owner);
   }
```

Bob is the owner of TxOrigin. Bob calls Eve's contract. Eve's contract calls TxOrigin and bypasses the **tx.origin** protection.

Recommendation: Do not use **tx.origin** for authorization.

Lines: Blizzard.sol#50

```
require(tx.origin == owner() || tx.origin == deployer, "!owner");
```

Low

1. Incorrect solidity version

   While immutable keyword was introduced only since solidity version 0.6.5, it’s incorrect to specify pragma solidity version as >0.6.0, because “solc” with version 0.6.24, for example, would not be able to compile it. Also, contracts aren’t using the visibility declaration for constructors, which was announced only since version **0.7.0**.

Recommendation: Please specify the correct solidity version in the pragma header.

2. Missing event for changing “devAllocation”

Changing critical values should be followed by the event emitting for better tracking off-chain.

Recommendation: Please emit events on the critical values changing.

Lines: Blizzard.sol#36-39

```
function setDevAllocation(uint256 _alloc) external onlyOwner {
   require(_alloc < MAX_ALLOCATION, "invalid value");
   devAllocation = _alloc;
}
```

3. Missing event for changing “devWallet”

Changing critical values should be followed by the event emitting for better tracking off-chain.

Recommendation: Please emit events on the critical values changing.

Lines: Blizzard.sol#41-43

```
function setDevWallet(address _wallet) external onlyOwner {
   devWallet = _wallet;
}
```

4. Missing event for “setMarketWallet”

Changing critical values should be followed by the event emitting for better tracking off-chain.

Recommendation: Please emit events on the critical values changing.

Lines: Blizzard.sol#45-47

```
function setMarketWallet(address _wallet) external onlyOwner {
   marketWallet = _wallet;
}
```

5. Missing event for “setMinter”

Changing critical values should be followed by the event emitting for better tracking off-chain.

Recommendation: Please emit events on the critical values changing.

Lines: Blizzard.sol#49-52

```
function setMinter(address _minter, bool _flag) external onlyOwner {
   require(tx.origin == owner() || tx.origin == deployer, "!owner");
   minters[_minter] = _flag;
}
```

6. State variables that could be declared constant

Constant state variables should be declared constant to save gas.

Recommendation: Add the **constant** attributes to state variables that never change.

7. Boolean equality

Boolean constants can be used directly and do not need to be compared to **true** or **false**.

Recommendation: Remove the equality to the boolean constant.

Lines: LPVault.sol#364

```
if (users.contains(_user) == true) {
```

Lines: LPVault.sol#370

```
if (users.contains(_user) == false) {
```

Lines: JoeLPStrategy.sol#135

```
if (enabledAutoCompound == true) {
```

Lines: JOEVault.sol#292

```
if (users.contains(_user) == true) {
```

Lines: JOEVault.sol#298

```
if (users.contains(_user) == false) {
```

Lines: Blizzard.sol#23

```
require(minters[msg.sender] == true, "!minter");
```

Lines: Blizzard.sol#55

```
if (mintersWithoutFee[msg.sender] == true) {
```

Lines: PangolinLPStrategy.sol#111

```
if (enabledAutoCompound == true) {
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **2** medium and **7** low severity issues.

Notice:

The QuickLPStrategyV3 contract implements the StratManager which is outside the scope of the current audit, so we cannot guarantee its functions and secureness.

Notice 2:

The QuickLPStrategyv4 contract implements the StratManager and FeeManager which is outside the scope of the current audit, so we cannot guarantee its functions and secureness.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.