HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Xtblock

Date: August 23rd, 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 Xtblock.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token locker
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/xtblock/xtt/tree/main/bep20
Commit	18c8dd3f5d52e53796002fc7a0406a07ee762759
Technical Documentation	NO
JS tests	NO
Timeline	20 AUGUST 2021 - 23 AUGUST 2021
Changelog	23 AUGUST 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 Xtblock (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 August 20th, 2021 - August 23rd, 2021.

Scope

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

Repository:
https://github.com/xtblock/xtt/tree/main/bep20
Commit:
18c8dd3f5d52e53796002fc7a0406a07ee762759
Technical Documentation: No
JS tests: No
Contracts:
BEP20-XTT-Contract.sol

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

Executive Summary

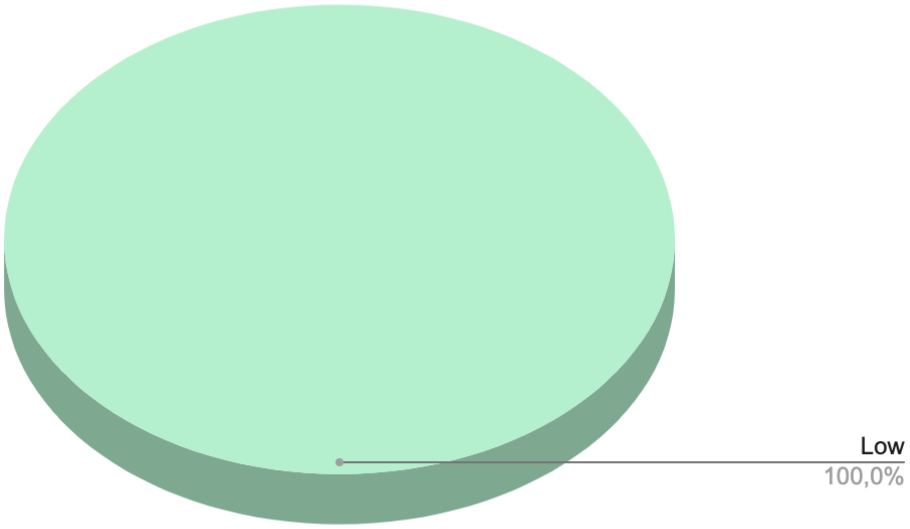
According to the assessment, the Customer's smart contracts are secured.

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 **6** low severity issues.

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

No medium severity issues were found.

Low

1. Incorrect versions of Solidity

   The pragma header specifies a solidity version starting from 0.6.0 and up to 0.8.0 excluding it. But actually, the contract could not be compiled because constructor visibility was removed only since 0.7.0, therefore you cannot compile the contract using the most common 0.6.12 solidity version.

   Recommendation: Please specify the strict version of solidity which were used to write code for. The recommended version is 0.7.6

Lines: XTT-TokenTimeLock.sol#3

```
pragma solidity >=0.6.0 <0.8.0;
```

2. Missing event on release time change

   Updating the release time for tokens is recommended to follow up with the emitting an event. This will make it easier to track the updates off-chain.

   Recommendation: Please emit the event on updating the release time.

Lines: XTT-TokenTimeLock.sol#70-75

```
function extendLockTime(uint256 newReleaseTime_) public virtual {
   require(_unlocker == msg.sender, "Ownable: caller is not the current
unlocker");
   require(newReleaseTime_ > releaseTime(), "TokenTimelock: new release
time can't be before the current release time");
   require(newReleaseTime_ <= releaseTime() + 365 days, "TokenTimelock: new
release time can't be longer than the current release time + 365 days");
   _releaseTime = newReleaseTime_;
}
```

3. Missing event on updating the unlocker

   Updating the unlocker address is recommended to follow up with the emitting an event. This will make it easier to track the updates off-chain.

   Recommendation: Please emit the event on updating the unlocker address.

Lines: XTT-TokenTimeLock.sol#80-83

```
function setUnlocker(address newUnlocker_) public virtual {
   require(_unlocker == msg.sender, "Ownable: caller is not the current
unlocker");
   _unlocker = newUnlocker_;
}
```

4. Missing zero-address validation

   In both places, constructor and setUnlocker(address) functions there is no check for zero address which could lead to mistakenly setting some of them to 0x0

   Recommendation: Please add a zero-check for addresses

5. A public function that could be declared external

   public functions that are never called by the contract should be declared **external** to save gas.

   Recommendation: Use the **external** attribute for functions never called from the contract.

Lines: XTT-TokenTimeLock.sol#70

```
function extendLockTime(uint256 newReleaseTime_) public virtual {
```

Lines: XTT-TokenTimeLock.sol#80

```
function setUnlocker(address newUnlocker_) public virtual {
```

Lines: XTT-TokenTimeLock.sol#88

```
function release(uint256 releasedValue_) public virtual {
```

6. Maximum line length

   The solidity provides style guides as well as code layout recommendations where they have a recommended maximum line length.

   Recommendation: Please follow the recommended maximum line length.

Lines: XTT-TokenTimeLock.sol#32, 72, 73, 90

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 **6** low severity issues.

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.