BoxFight

Game&Token

SMART CONTRACT AUDIT

15.10.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer.......... 3
2. About the Project and Company.......... 4
2.1 Project Overview.......... 5
3. Vulnerability & Risk Level.......... 6
4. Auditing Strategy and Techniques Applied.......... 7
4.1 Methodology.......... 7
4.2 Used Code from other Frameworks/Smart Contracts.......... 8
4.3 Tested Contract Files.......... 9
4.4 Metrics / CallGraph.......... 10
4.5 Metrics / Source Lines.......... 11
4.6 Metrics / Capabilities.......... 12
4.7 Metrics / Source Unites in Scope.......... 13
5. Scope of Work.......... 14
5.1 Manual and Automated Vulnerability Test.......... 15
5.2. SWC Attacks & Special Checks.......... 16
7. Verify Claims.......... 20
8. Executive Summary.......... 22
9. Deployed Smart Contract.......... 22

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement ofBoxFight Token. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (02.06.2021)	Layout
0.5 (03.06.2021)	Automated Security Testing Manual Security Testing
0.8 (03.06.2021)	Testing SWC Checks
0.9 (04.06.2021)	Summary and Recommendation
1.0 (05.06.2021)	Final document

2. About the Project

Website:www.boxfight.monster

GitHub:https://github.com/BoxFightOffical/BoxFight

Twitter: https://twitter.com/fight4box

Telegram: https://t.me/BoxFight_CN

BSCScan (BoxFight Token):https://bscscan.com/address/0xFfF333DC397A3EDFBCb9926B9Dc7E8D43C93524F

2.1 Project Overview

BoxFight is a unique platform that combines the best tokenomics of currrent frictionless yield protocols for instant rewards with the additional benefits of staking in our upcoming marketplace. This way the best rewards can be guaranteed without any token inflation. A 3% transaction tax goes to holders (later on merchants too), stakers, and a perpetual marketing and development fund. This project is built to keep going and continually expand further until it has its own ecosystem to call its own. The $BoxFight system guarantees token rewards to LP stakers on every block, regardless if there was a $BoxFight transaction on it or not. Under the same system, rewards will scale as the project grows, whilst ensuring the rewards pool can never run out.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

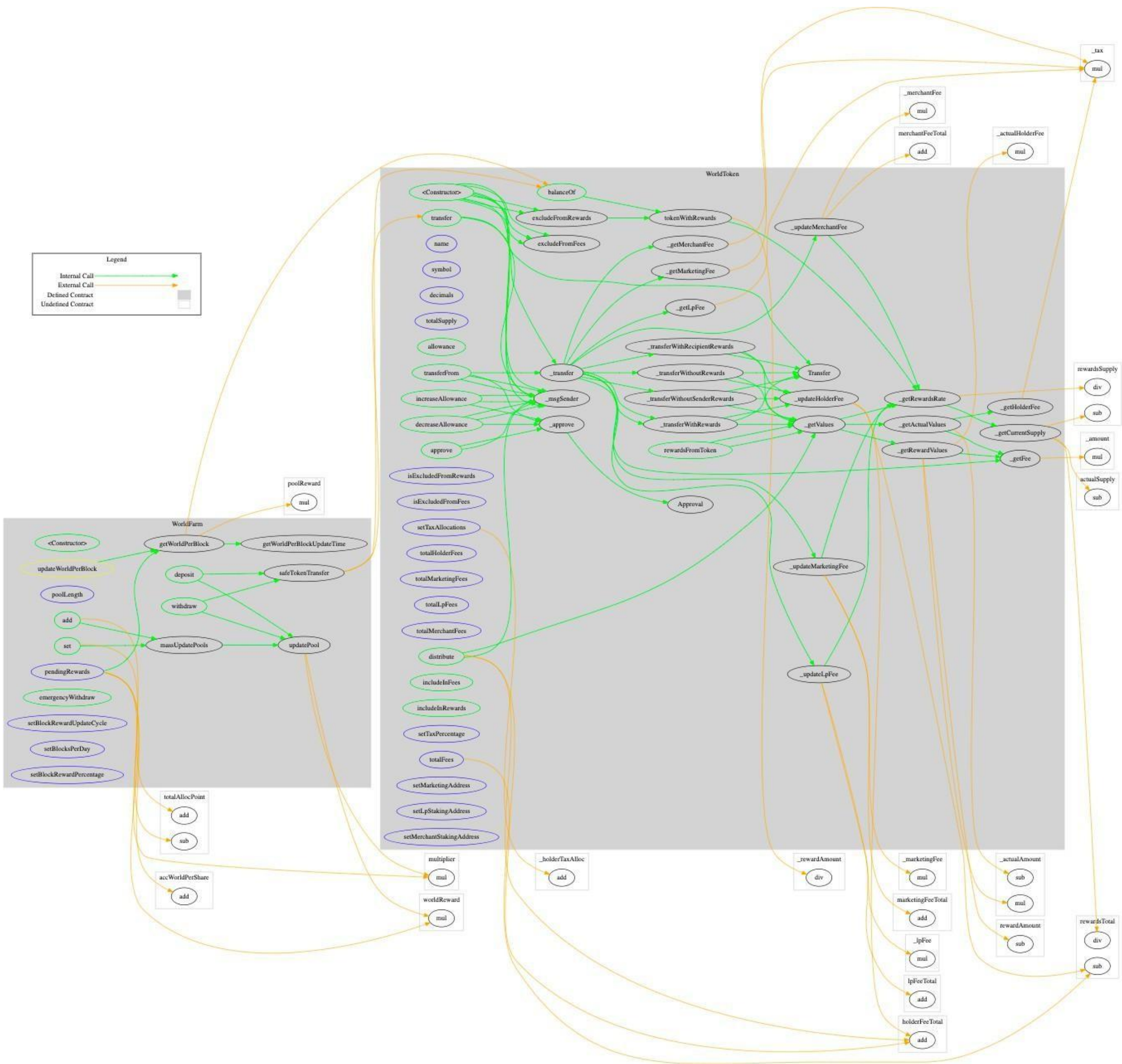
1. SafeMath.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0/contracts/math/SafeMath.sol
2. BEP20.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0/contracts/token/BEP20/BEP20.sol
3. SafeBEP20.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0/contracts/token/BEP20/SafeBEP20.sol
4. Ownable.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0/contracts/access/Ownable.sol
5. Address.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0/contracts/utils/Address.sol
6. Context.sol (0.6.0)
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.3.0/contracts/GSN/Context.sol

4.3 Tested Contract Files

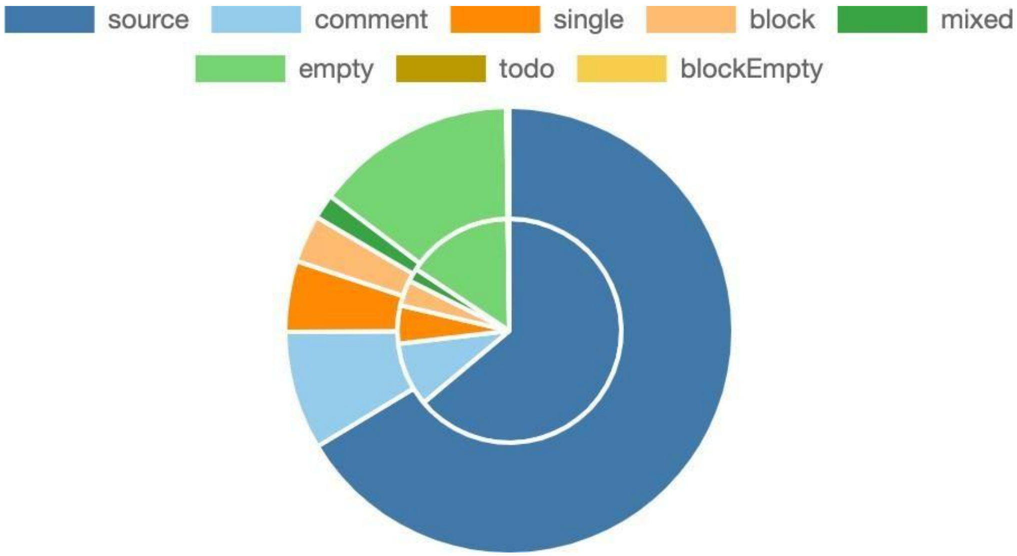
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (SHA256)
BoxFightGame.sol	01b1e800e02d36aadbfb13b6ba4e6fbe
BoxFightToken.sol	49c943c28ecd903153bad9fbb1c7d340

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines



4.6 Metrics / Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`0.7.4`			**** (0 asm blocks)	

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
`yes`					

Public	Payable
44	0

External	Internal	Private	Pure	View
21	39	20	4	25

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	BoxFight-main/contracts/BoxFightGame. sol	1		272	264	189	48	151	
	BoxFight-main/contracts/BoxFightToken.sol	1		565	502	377	34	289	
	Totals	**2**		**837**	**766**	**566**	**82**	**440**	

5. Scope of Work

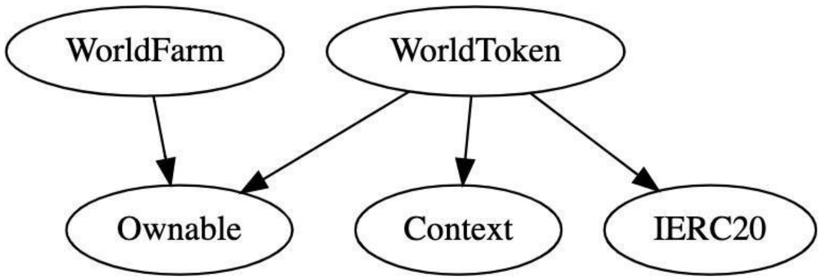
TheBoxFightToken Team provided us with the files that needs to be tested. The scope of the audit are the Game and Token contracts. Following

contracts with the direct imports been tested
BoxFightGame.
solBoxFightTo
ken.sol

The team put forward the following assumptions regarding the security, usage of the contracts:

- LP Token Staker are always able to withdraw LP Token shares
- BoxFight Token deployer cannot mint any new token
- BoxFight Token deployer cannot burn or lock user funds
- BoxFight Token deployer cannot pause the contract
- Checking the overall security of the contracts

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES

During the audit, Chainsulting's experts found **no Low issues** in the code of the smart contract.

5.2. SWC Attacks & Special Checks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	

7. Verify Claims

7.1 LP Token Staker are always able to withdraw LP Token shares ✅

Status: tested and verified
Code: Ln 217 – 246BoxFightGame.sol

```
function withdraw(uint256 _pid, uint256 _amount) public
    { PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    require(user.amount >= _amount, "Withdraw amount is greater than user amount");

    updatePool(_pid);

    uint256 pending =
    user.amount.mul(pool.accBoxFightPerShare).div(1e12).sub(user.rewa
    rdDebt); if (pending > 0) {
        safeTokenTransfer(msg.sender, pending);
    }
    if (_amount > 0) {
        user.amount = user.amount.sub(_amount);
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
    }
    user.rewardDebt =
    user.amount.mul(pool.accBoxFightPerShare).div(1e12); emit
    Withdraw(msg.sender, _pid, _amount);
}
function emergencyWithdraw(uint256 _pid) public
    { PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    user.amount = 0;
    user.rewardDebt = 0;

    pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
}
```

7.2 BoxFight Token deployer cannot mint any new token Status: tested and verified ✅
Code: Ln 62 BoxFightToken.sol

```
uint256 private constant ACTUAL_TOTAL = 100_000_000 * 1e18;
```

7.3 BoxFight Token deployer cannot pause the contract Status: tested and verified ✅
Code:BoxFightToken.sol

7.4 BoxFight Token deployer cannot burn or lock user funds Status: tested and verified ✅
Code:BoxFightToken.sol

7.5 Checking the overall security of the contracts ✅

8. Executive Summary

The overall code quality of the project is very good, not overloaded with unnecessary functions, these is greatly benefiting the security of the contract. It correctly implemented widely-used and reviewed contracts from OpenZeppelin and for safe mathematical operations.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no issues were found after the manual and automated security testing.

9. Deployed Smart Contract

VERIFIED

BoxFight Token
https://bscscan.com/address/0xFfF333DC397A3EDFBCb9926B9Dc7E8D43C9
3524F