Oiler Staking

SMART CONTRACT AUDIT

13.05.2021

Made in Germany by Chainsulting.de

Table of contents

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 of Oiler DeFi. 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 (19.04.2021)	Layout
0.5 (20.04.2021)	Verify Claims and Test Deployment
0.6 (21.04.2021)	Testing SWC Checks
0.8 (22.04.2021)	Automated Security Testing Manual Security Testing
0.9 (22.04.2021)	Summary and Recommendation
1.0 (23.04.2021)	Final document
1.1 (13.05.2021)	Added deployed contract

2. About the Project and Company

Company address:

Oiler DeFi c/o International Corporation Services Ltd
PO Box 472, Harbour Place 2nd
Floor103 South Church Street
Grand Cayman
KY1-1106, Cayman Islands

Website: https://oiler.network

Twitter: https://twitter.com/OilerNetwork

Telegram: https://t.me/oilernetwork

Telegram: https://t.me/oiler_official

Medium: https://medium.com/oiler-network

LinkedIn: https://www.linkedin.com/company/oiler-network/

2.1 Project Overview

Oiler is a protocol for blockchain native derivatives.

Before 2019/2020, it was practically impossible to deliver blockchain native derivatives. Without stablecoins and AMMs (automatic market makers), it was not possible to provide a reliable pricing solution. What has changed? Stablecoins introduced a non-volatile on-chain base for pricing (a USD peg) and AMMs introduced a pricing discovery mechanism; on-chain and with high volumes. It has also been proven that if the markets are efficient then the AMMs are efficient too and arbitrageurs will set the price right.

In order to settle derivatives on-chain nowadays, we need to ensure that the payout can be calculated entirely on-chain. At Oiler, they not only assume that they will not take off-chain data but also that there is no oracle hidden behind the layers of on-chain data sources that the smart contracts use. It means that the prices of the underlying instruments should not be derived from a protocol that uses off-chain oracles. Moreover, it is desirable to avoid any on-chain oracles like Uniswap since they can always be manipulated within a flash loan based attack. The last requirement is much stronger but still holds for the initial set of Oiler products.

DeFi users are most likely to use Oiler if they already have exposure to blockchain protocol parameters — their operations rely on big shifts in the network behavior, manifesting via big hashrate changes, massive gas price movements and protocol behavior changes. Who would that be?

- Exchanges that cover the highly volatile on-chain assets withdrawal costs
- Miners having exposure to shifting block reward and transaction fees
- Institutions with exposures to many blockchains and a need for hedging the protocol-level risks

Oiler will continue bringing products — both new underlying blockchain parameters and new instrument types that will be related to the category of blockchain protocol trading.They will continue working on pricing models without oracles and will look at the instruments related to both Ethereum 1.0 and Ethereum 2.0 (and their coexistence).

read more at https://docs.oiler.network/oiler-network

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)

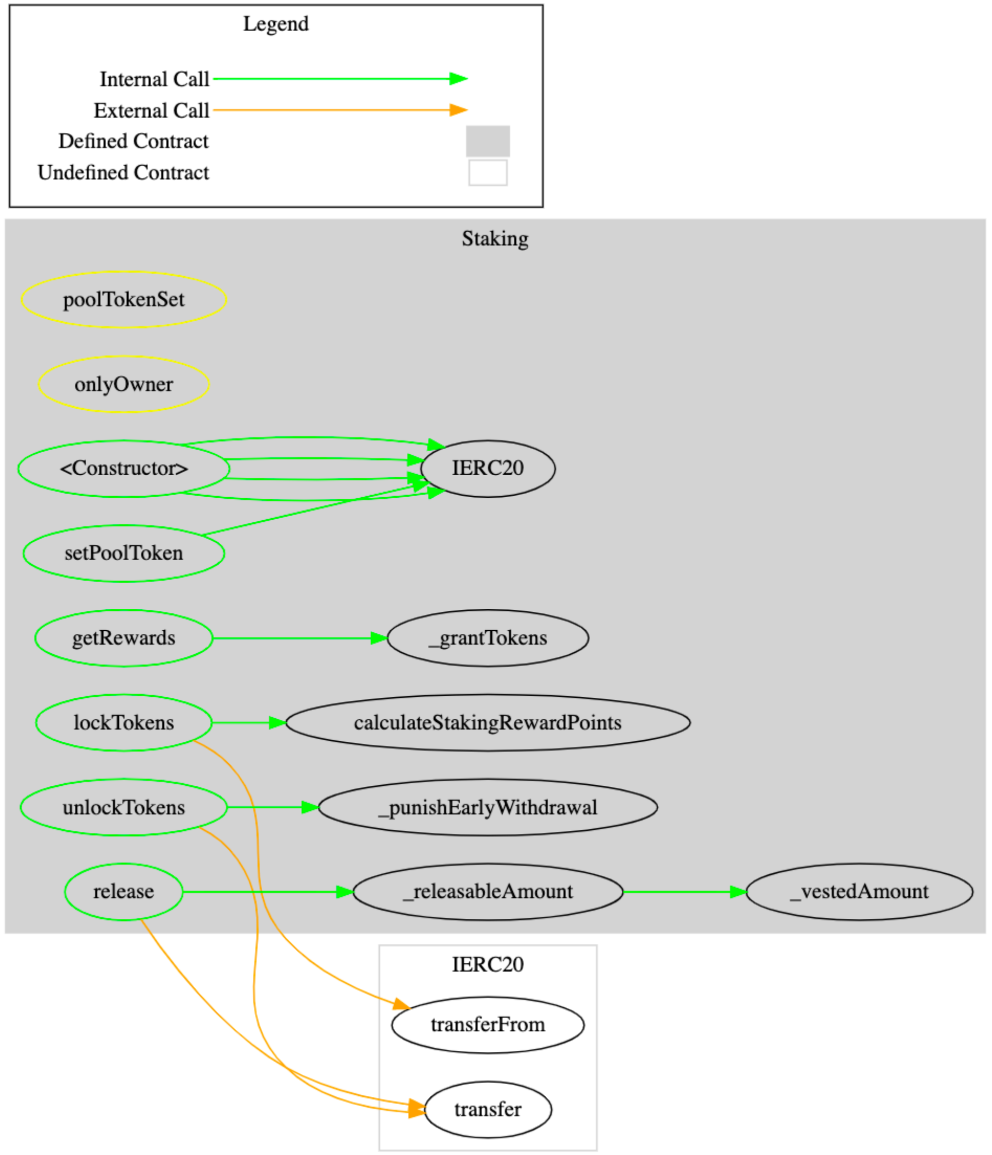
Dependency / Import Path	Link / Source
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.0.0/contracts/token/ERC20/IERC20.sol

4.3 Tested Contract Files

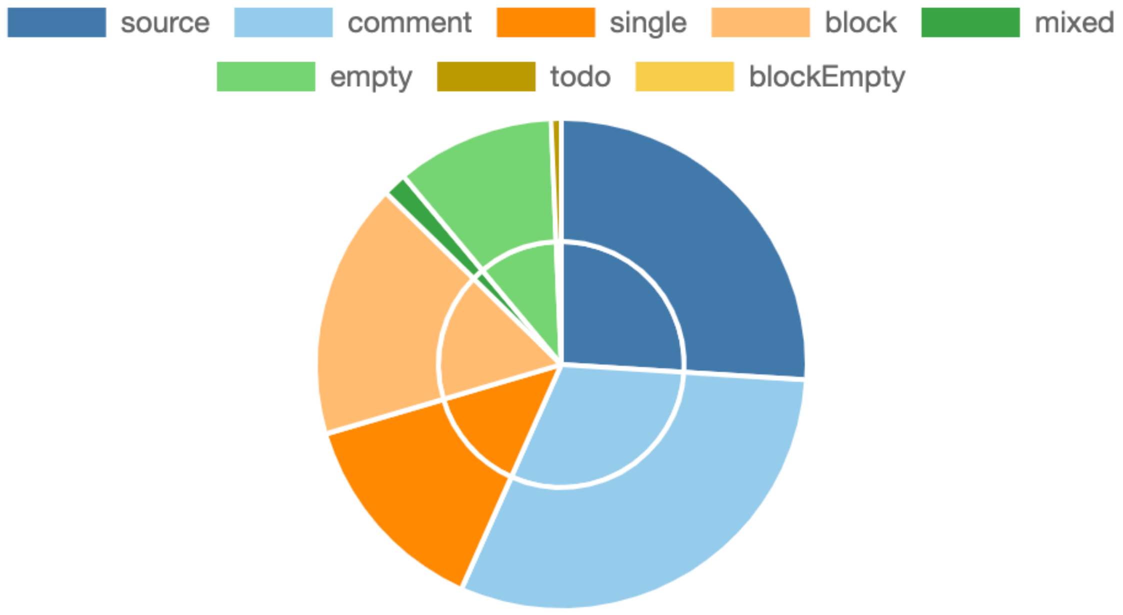
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
Staking.sol	31c642767700aff93c615a7eeba3d878

4.4 Metrics / CallGrap



4.5 Metrics / Source Lines



4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.3`			**** (0 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`					

🌐Public	💰Payable
6	0

External	Internal	Private	Pure	View
0	12	0	1	2

StateVariables

Total	🌐Public
11	11

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/Staking.sol	1		295	295	117	138	92	📤
📝	**Totals**	**1**		**295**	**295**	**117**	**138**	**92**	📤

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

5. Scope of Work

The Oiler Team provided us with the files that needs to be tested. The scope of the audit is the Oiler Staking contract.

Following contracts with the direct imports has been tested:

- Staking.sol

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

- Owner / deployer is not able to withdraw staked amounts of users
- No one can stake anymore, after the staking program has ended
- It’s not possible to claim rewards before the staking program ends
- No one else without the owner of the staked amount can withdraw stake or reward
- Check overall security and business logic

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

5.1.1 A floating pragma is set
Severity: LOW
Status: FIXED
File(s) affected: Staking.sol

Attack / Description	**Code Snippet**	**Result/Recommendation**
The current pragma Solidity directive is ^0.8.3; It is recommended to specify a fixed version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.	Line 1 pragma solidity ^0.8.3;	It is recommended to follow the example (0.8.3), as future compiler versions may handle certain language constructions in a way the developer did need foresee. Not effecting the overall contract functionality.

5.1.2 Unsecure calculations
Severity: LOW
Status: FIXED (0.8.4)
File(s) affected: Staking.sol

Attack / Description	Code Snippet	Result/Recommendation

The calculations could cause unhandled errors such as integer overflow.	Line 145
	`uint256 stakingRewardPoints = uint256(tokenAmount_) * uint256(lockingPeriodInBlocks_) * uint256(lockingPeriodInBlocks_);`
	
	Line 235
	`uint256 amountEarned = stakingFundAmount * rewardPointsEarned[msg.sender] / totalRewardPoints;`
		We recommend using the OpenZeppelin library SafeMath to avoid unsecure calculations.

5.2. SWC

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	X
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	✓

6. Test Deployment

6.1.1 Deployment USDC ERC20-Token

Tx: https://kovan.etherscan.io/tx/0x24225ba2c42fcdd9fe1c7c26829242489f5945263713f34625680d272ea933dc
Contract: https://kovan.etherscan.io/address/0x46afd69cb9843099e6bcb18a0baa0cf70e79775d

6.1.2 Deployment Oiler ERC20-Token

Tx: https://kovan.etherscan.io/tx/0x7db95edd3e5aed7c83cdc5fb829ca2afc92c3949880957f1cc6e4dfd61f2aca5
Contrac: https://kovan.etherscan.io/address/0x42b313c3684aeff961ed4163bdfb9906edd5fac5

6.1.3 Create liquidity pool token (UNI-V2)

Tx: https://kovan.etherscan.io/tx/0x72f36e77d50886125a03c4fbd1c8508b23ef70b124f21d72430dc0766ecdfe15
Contract: https://kovan.etherscan.io/address/0x001a718eb6ac09d886139dd14a7e02af577fd930

6.1.4 Deployment staking contract

Tx: https://kovan.etherscan.io/tx/0x2282c3eef0688413260f6310728978274e9a2d86905e6ad50f8aa8e6c2d0b1b3
Contract: https://kovan.etherscan.io/address/0xee29c7623893ba46a5d7879c360d6943e1e8e06f

6.2 Verify Claims

6.2.1 Owner / deployer is not able to withdraw staked amounts of users

By staking tokens, the stake is correctly mapped to the message sender (function caller). To withdraw stake the user has to call the *unlockTokens* method. This method restricts withdraws to the staked amount mapped to the senders address. In addition there is no other functionality to withdraw users stake. Thus users stakes cannot be withdrawn by the owner of the contract. The only privilege of the owner is to make the contract running by calling *setPoolToken* once.

```
Stake memory stake = stakes[msg.sender];
uint256 stakeAmount = stake.tokenAmount;
require(poolToken.transfer(msg.sender, stakeAmount), "Pool token transfer failed");
```

6.2.2 No one can stake anymore, after the staking program has ended

The l*ockTokens* method requires the current block to be before the end block of staking program. In addition there is no other functionality to stake tokens in the smart contract. Thus it is not possible to lock tokens if the end block of staking program is reached.

```
require(block.number <= stakingProgramEndsBlock - lockingPeriodInBlocks_, "Your lock period exceeds Staking Program duration");
```

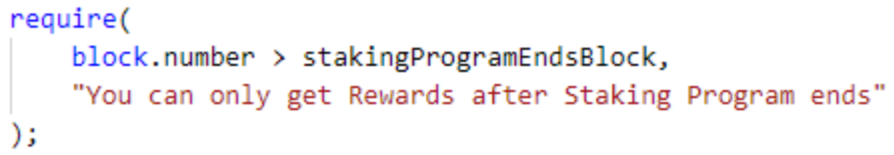
Function call of *lockTokens* after staking programs has ended fails as expected.

```
Fail with error 'Your lock period exceeds Staking Program duration'
```

Tx: https://kovan.etherscan.io/tx/0xcb22a03d4b8b958f9ea5d7775e5c5bf2c2eb11d3537f210571f1acda4f16be24

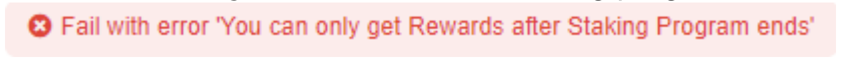
6.2.3 It is not possible to claim rewards before the staking program ends

The *getRewards* method requires the current block number to be greater than the end block number of staking program. In addition there is no other functionality in the contract to claim rewards. Thus it is not possible to claim rewards before the staking program ends.



```
require(
    block.number > stakingProgramEndsBlock,
    "You can only get Rewards after Staking Program ends"
);
```

Function call of *getRewards* before staking programs has ended fails as expected.



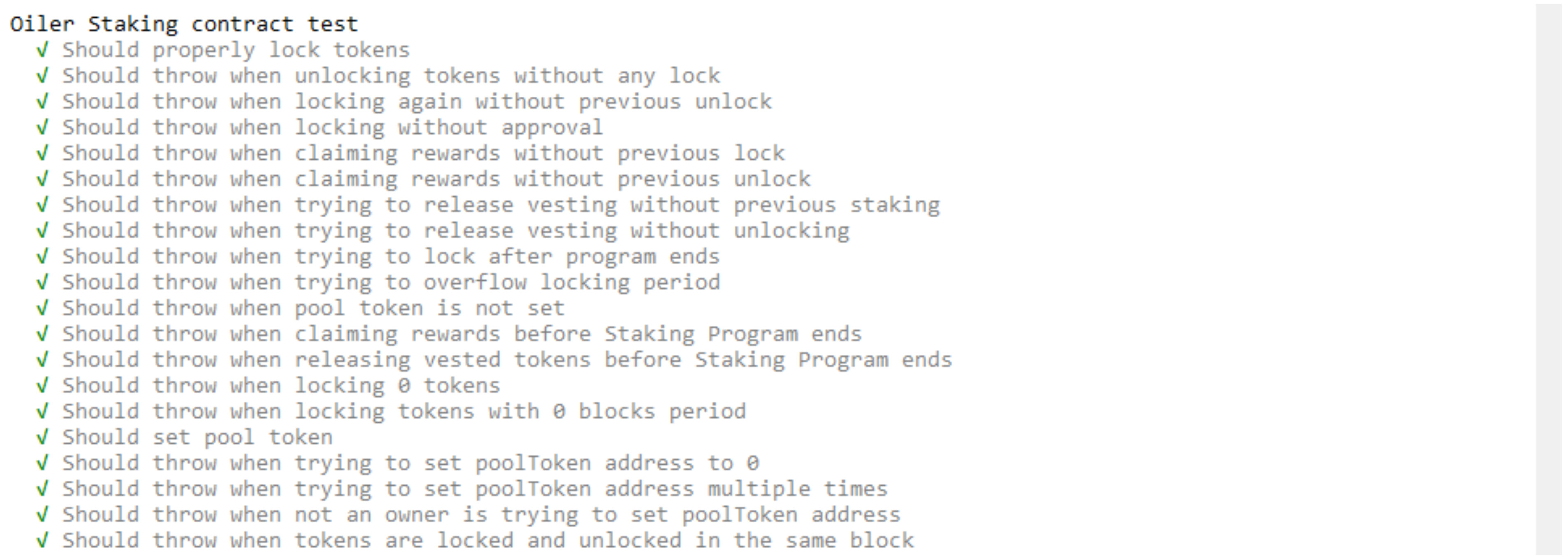
```
Fail with error 'You can only get Rewards after Staking Program ends'
```

Tx: https://kovan.etherscan.io/tx/0x1bcb735885e7ed897206cc74010c9eb9a4d78825168a883d44bbf1a859959db4

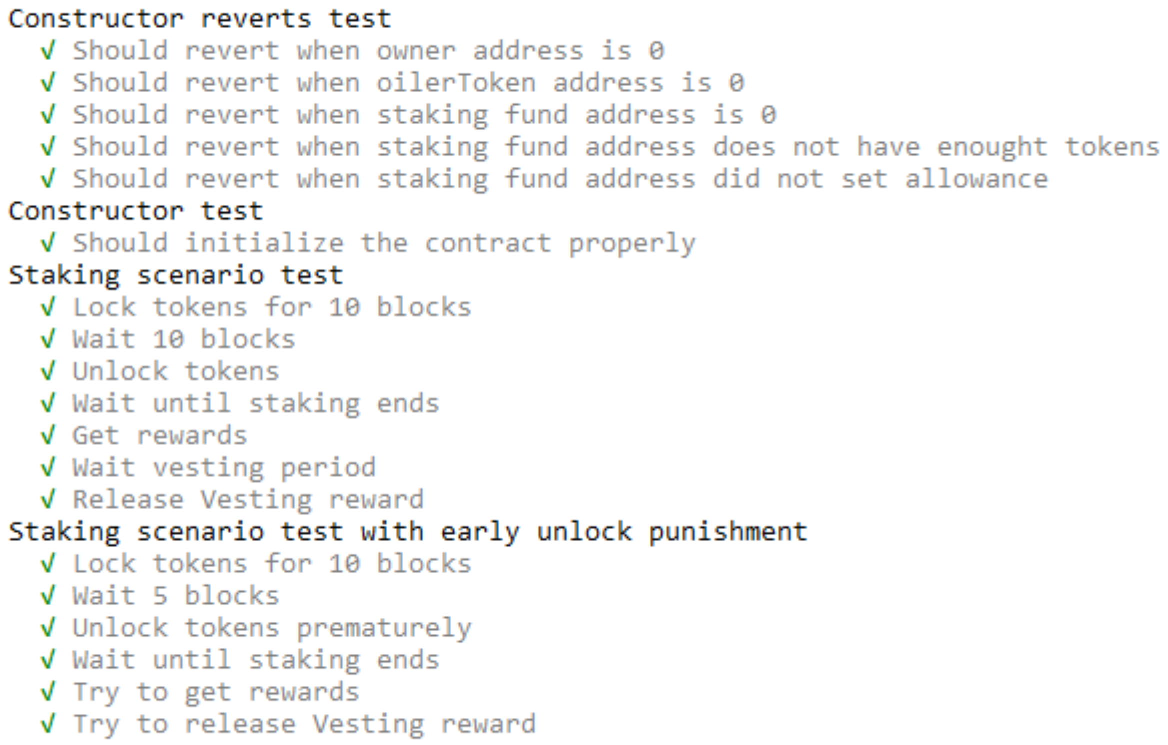
6.2.4 No one else without the owner of the staked amount can withdraw stake or reward

By calling *lockTokens* the token amount is correctly mapped to the calling user. The only way to withdraw that stake is to call unlockTokens. This function withdraws only the stake of the message sender (who called the function) to his address. Thus it is not possible for users to withdraw stakes not belonging to their address. The rewards of staking are mapped to the users in the same way and can also only be claimed by its owner. It is not possible to claim rewards not belonging to callers address.

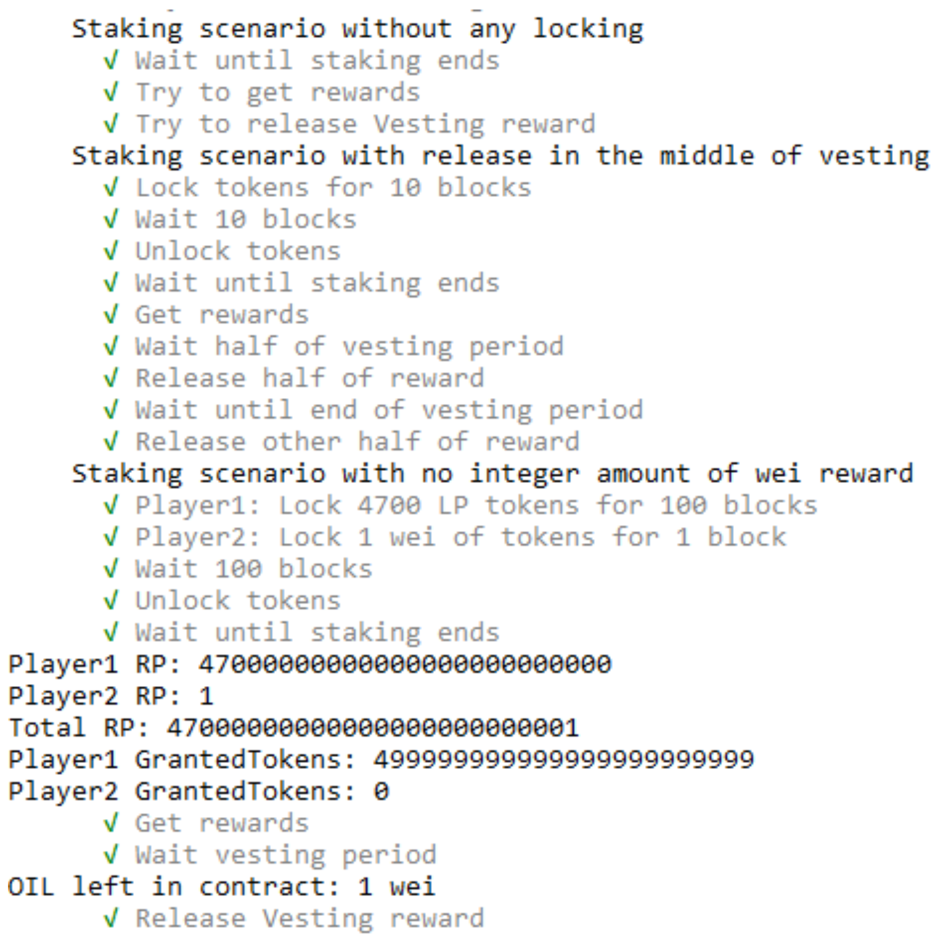
6.3 Unit Test



```
Oiler Staking contract test
  √ Should properly lock tokens
  √ Should throw when unlocking tokens without any lock
  √ Should throw when locking again without previous unlock
  √ Should throw when locking without approval
  √ Should throw when claiming rewards without previous lock
  √ Should throw when claiming rewards without previous unlock
  √ Should throw when trying to release vesting without previous staking
  √ Should throw when trying to release vesting without unlocking
  √ Should throw when trying to lock after program ends
  √ Should throw when trying to overflow locking period
  √ Should throw when pool token is not set
  √ Should throw when claiming rewards before Staking Program ends
  √ Should throw when releasing vested tokens before Staking Program ends
  √ Should throw when locking 0 tokens
  √ Should throw when locking tokens with 0 blocks period
  √ Should set pool token
  √ Should throw when trying to set poolToken address to 0
  √ Should throw when trying to set poolToken address multiple times
  √ Should throw when not an owner is trying to set poolToken address
  √ Should throw when tokens are locked and unlocked in the same block
```



```
Constructor reverts test
  √ Should revert when owner address is 0
  √ Should revert when oilerToken address is 0
  √ Should revert when staking fund address is 0
  √ Should revert when staking fund address does not have enought tokens
  √ Should revert when staking fund address did not set allowance
Constructor test
  √ Should initialize the contract properly
Staking scenario test
  √ Lock tokens for 10 blocks
  √ Wait 10 blocks
  √ Unlock tokens
  √ Wait until staking ends
  √ Get rewards
  √ Wait vesting period
  √ Release Vesting reward
Staking scenario test with early unlock punishment
  √ Lock tokens for 10 blocks
  √ Wait 5 blocks
  √ Unlock tokens prematurely
  √ Wait until staking ends
  √ Try to get rewards
  √ Try to release Vesting reward
```



```
    Staking scenario without any locking
      √ Wait until staking ends
      √ Try to get rewards
      √ Try to release Vesting reward
    Staking scenario with release in the middle of vesting
      √ Lock tokens for 10 blocks
      √ Wait 10 blocks
      √ Unlock tokens
      √ Wait until staking ends
      √ Get rewards
      √ Wait half of vesting period
      √ Release half of reward
      √ Wait until end of vesting period
      √ Release other half of reward
    Staking scenario with no integer amount of wei reward
      √ Player1: Lock 4700 LP tokens for 100 blocks
      √ Player2: Lock 1 wei of tokens for 1 block
      √ Wait 100 blocks
      √ Unlock tokens
      √ Wait until staking ends
Player1 RP: 47000000000000000000000000
Player2 RP: 1
Total RP: 47000000000000000000000001
Player1 GrantedTokens: 499999999999999999999999
Player2 GrantedTokens: 0
      √ Get rewards
      √ Wait vesting period
OIL left in contract: 1 wei
      √ Release Vesting reward
```

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The overall code quality of the project is good, and the simplicity greatly benefits the overall security.

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 critical issues were found after the manual and automated security testing. We recommend to use SafeMath OpenZeppelin library for safe mathematical operations within the contract.

8. Deployed Smart Contract

VERIFIED

Smart Contract is deployed here:
https://etherscan.io/address/0xe546f8f17aff17c05dac9f9b4f9957f725fab087#code