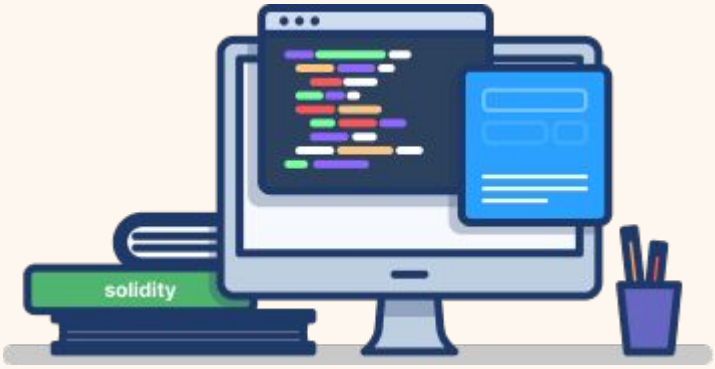
Coinbae Audit



Barnbridge Staking Pool Audit January 2021

Contents

Introduction

Audit:

In January 2021 Coinbae's audit report division performed an audit for the Barnbridge Staking Contract.

https://etherscan.io/address/0xb0fa2beee3cf36a7ac7e99b885b48538ab364853#code

Barnbridge:

BarnBridge is the first tokenized risk protocol. Before the advent of smart contract technology it was close to impossible to track & attribute yield to a divided allotment of capital, trustlessly & transparently, to provide hedges against any and all fluctuations. Conceptually, you can build derivative products from any type of market driven fluctuation to hedge various risks. Examples include, but are not limited to, interest rate sensitivity, fluctuations in underlying market price, fluctuations in predictive market odds, fluctuations in default rates across mortgages, fluctuations in commodity prices, and a seemingly infinite number of market based fluctuations to hedge a particular position.

As described in Barnbridge's whitepaper.

Introduction

Overview:

Information:
Name: Barnbridge Staking Pool

Pool, Asset or Contract address:
https://etherscan.io/address/0xb0fa2beee3cf36a7ac7e99b885b48538ab364853#code

Supply:
Current: 1,044,486 BOND

Explorers:
https://etherscan.io/address/0xb0fa2beee3cf36a7ac7e99b885b48538ab364853#code

Websites:
https://barnbridge.com/

Links:
Github

Introduction

Compiler related issues:

It is best practice to use the latest version of the solidity compiler supported by the toolset you use. This so it includes all the latest bug fixes of the solidity compiler. When you use for instance the openzeppelin contracts in your code the solidity version you should use should be 0.8.0 because this is the latest version supported.

Caution:

The solidity versions used for the audited contracts can be 0.6.0 --> 0.8.0 these versions have for instance the following known bugs so the compiled contract might be susceptible to:

EmptyByteArrayCopy – Medium risk

Copying an empty byte array (or string) from memory or calldata to storage can result in data corruption if the target array's length is increased subsequently without storing new data.
https://etherscan.io/solcbuginfo?a=EmptyByteArrayCopy

DynamicArrayCleanup – Medium risk

When assigning a dynamically-sized array with types of size at most 16 bytes in storage causing the assigned array to shrink, some parts of deleted slots were not zeroed out.
https://etherscan.io/solcbuginfo?a=DynamicArrayCleanup

Advice:

Update the contracts to the latest supported version of solidity by your contract. And set it as a fixed parameter not a floating pragma.

Audit Report Scope

Assertions and Property Checking:

1. Solidity assert violation.
2. Solidity AssertionFailed event.

ERC Standards:

1. Incorrect ERC20 implementation.

Solidity Coding Best Practices:

1. Outdated compiler version.
2. No or floating compiler version set.
3. Use of right-to-left-override control character.
4. Shadowing of built-in symbol.
5. Incorrect constructor name.
6. State variable shadows another state variable.
7. Local variable shadows a state variable.
8. Function parameter shadows a state variable.
9. Named return value shadows a state variable.
10. Unary operation without effect Solidity code analysis.
11. Unary operation directly after assignment.
12. Unused state variable.
13. Unused local variable.
14. Function visibility is not set.
15. State variable visibility is not set.
16. Use of deprecated functions: call code(), sha3(), ...
17. Use of deprecated global variables (msg.gas, ...).
18. Use of deprecated keywords (throw, var).
19. Incorrect function state mutability.
20. Does the code conform to the Solidity styleguide.

Convert code to conform Solidity styleguide:

1. Convert all code so that it is structured accordingly the Solidity styleguide.

Audit Report Scope

Categories:

High Severity:

High severity issues opens the contract up for exploitation from malicious actors. We do not recommend deploying contracts with high severity issues.

Medium Severity Issues:

Medium severity issues are errors found in contracts that hampers the effectiveness of the contract and may cause outcomes when interacting with the contract. It is still recommended to fix these issues.

Low Severity Issues:

Low severity issues are warning of minor impact on the overall integrity of the contract. These can be fixed with less urgency.

Audit Report

10	10	0
Identified	Confirmed	Critical

0	6	4
High	Medium	Low

Analysis:
https://etherscan.io/address/0xb0fa2beee3cf36a7ac7e99b885b48538ab364853#code

Risk:
Low

Medium severity: Coding best practices

Function could be marked as external SWC-000:

Calling each function, we can see that the public function uses 496 gas, while the external function uses only 261. The difference is because in public functions, Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Memory allocation is expensive, whereas reading from calldata is cheap. So if you can, use external instead of public.

Affected lines:

1. function deposit(address tokenAddress, uint256 amount) public nonReentrant { [#57]
2. function withdraw(address tokenAddress, uint256 amount) public nonReentrant { [#148]
3. function emergencyWithdraw(address tokenAddress) public { [#257]
4. function balanceOf(address user, address token) public view returns (uint256) { [#308]
5. function getEpochPoolSize(address tokenAddress, uint128 epochId) public view returns (uint256) { [#326]

Medium severity: Coding best practices

DoS With Block Gas Limit (SWC-128)

When smart contracts are deployed or functions inside them are called, the execution of these actions always requires a certain amount of gas, based of how much computation is needed to complete them. The Ethereum network specifies a block gas limit and the sum of all transactions included in a block can not exceed the threshold.

Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. Modifying an array of unknown size, that increases in size over time, can lead to such a Denial of Service condition.

Affected lines:

1. while (max > min) { [#293]

Low severity: Solidity style guide naming convention issues found

A floating pragma is set SWC-103:

The current pragma Solidity directive is ""^0.6.12"". It is recommended to specify a fixed compiler 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.

Affected lines:

1. pragma solidity ^0.6.0; [#2]

Requirement violation SWC-123:

The Solidity require() construct is meant to validate external inputs of a function. In most cases, such external inputs are provided by callers, but they may also be returned by callees. In the former case, we refer to them as precondition violations.

Affected lines:

1. contract Staking is ReentrancyGuard { [#8]
2. uint256 allowance = token.allowance(msg.sender, address(this)); [#61]

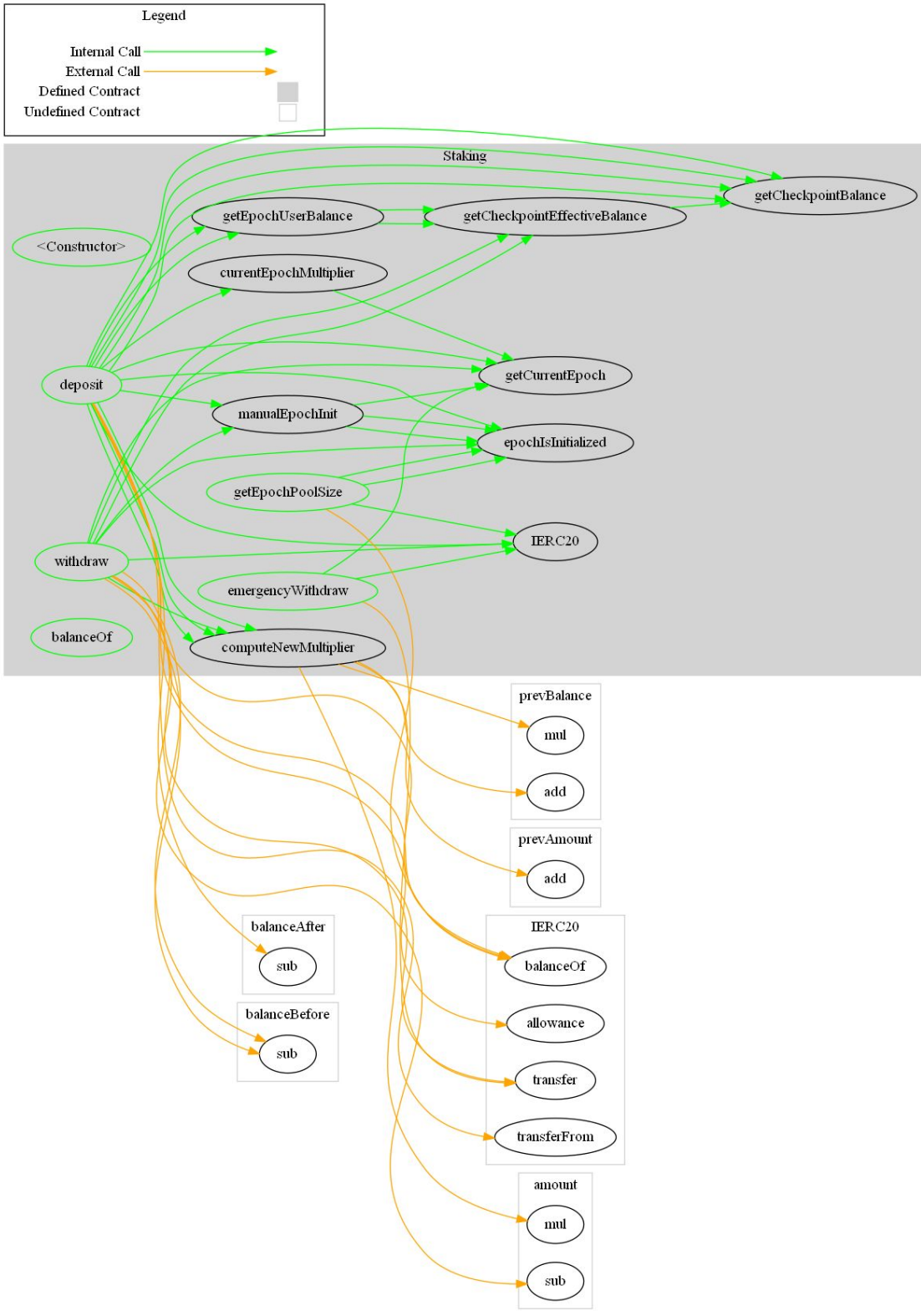
Assert violation SWC-110:

The Solidity assert() function is meant to assert invariants. Properly functioning code should *never* reach a failing assert statement.

Affected lines:

1. return uint128((block.timestamp - epoch1Start) / epochDuration + 1); [#320]

Contract Flow



Audit Team

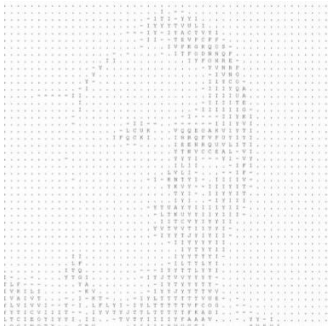
Team Lead: Eelko Neven

Eelko has been in the it/security space since 1991. His passion started when he was confronted with a formatted hard drive and no tools to undo it. At that point he started reading a lot of material on how computers work and how to make them work for others. After struggling for a few weeks he finally wrote his first HD data recovery program. Ever since then when he was faced with a challenge he just persisted until he had a solution.

This mindset helped him tremendously in the security space. He found several vulnerabilities in large corporation servers and notified these corporations in a responsible manner. Among those are Google, Twitter, General Electrics etc.

For the last 12 years he has been working as a professional security /code auditor and performed over 1500 security audits / code reviews, he also wrote a similar amount of reports.

He has extensive knowledge of the Solidity programming language and this is why he loves to do Defi and other smartcontract reviews.



Email:
info@coinbae.com

Conclusion

We performed the procedures as laid out in the scope of the audit and there were 10 findings, 6 medium and 4 low. The medium risk issues do not pose a security risk as they are best practice issues that is why the overall risk level is low.

Disclaimer

Coinbae audit is not a security warranty, investment advice, or an endorsement of the Barnbridge protocol. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the the Barnbridge protocol team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.