CERTIK

Shared Stake

DAO

Security Assessment

March 29th, 2021

Audited By:
Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org
Reviewed By:
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Shared Stake - DAO
Description	A DAO, staking system, and yield bearing wrapper token
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 03e977f343ccf8507451a8728984ecc248a6d7fe

Audit Summary

Delivery Date	March 29th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	March 9th, 2021 - March 29th, 2021

Vulnerability Summary

Total Issues	19
Total Critical	0
Total Major	0
Total Medium	1
Total Minor	7
Total Informational	11

Executive Summary

The report represents the results of our engagement with Shared Stake on their implementation of their DAO smart contracts.

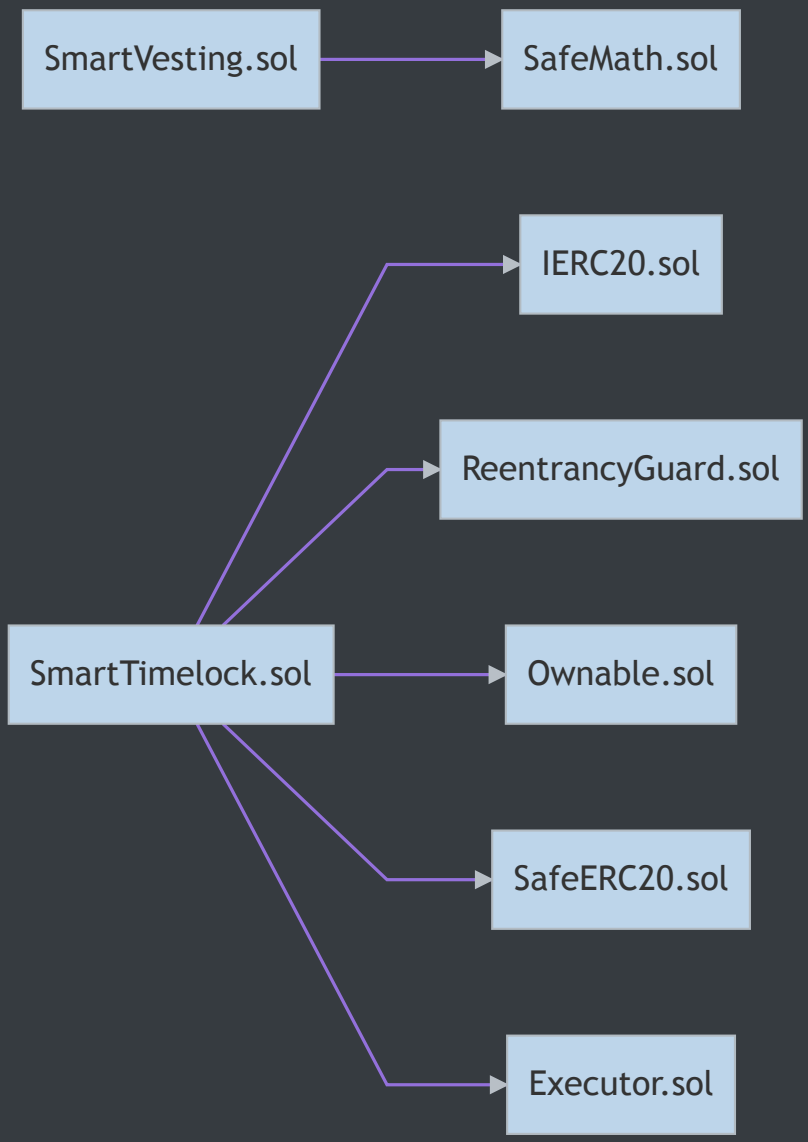
Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract's safety.

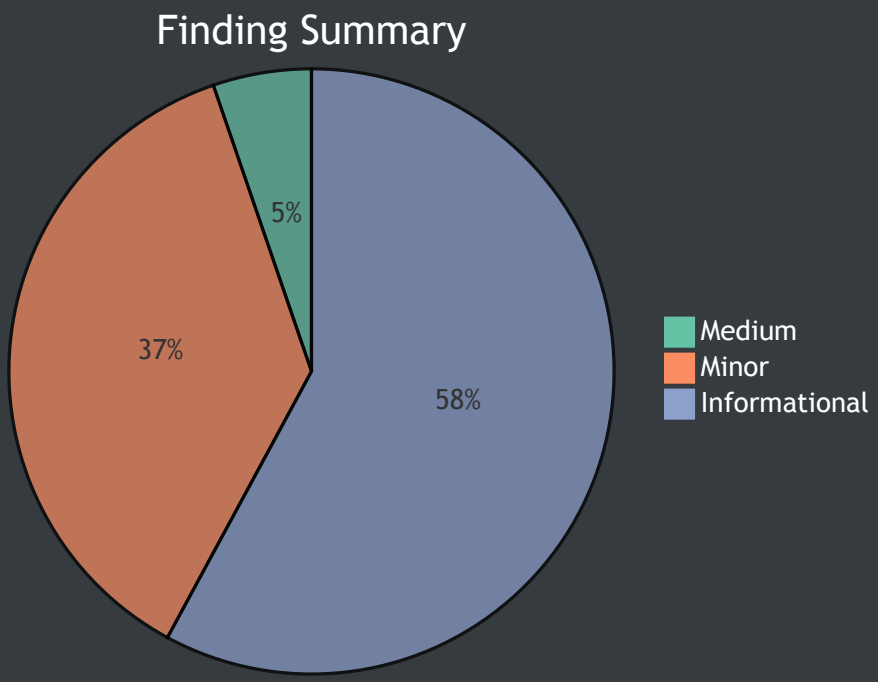
The minters of the system can arbirtarily burn tokens, increasing its centralization.

Files In Scope

ID	Contract	Location
AIR	Airdrop_v2.sol	Airdrop_v2.sol
MIN	Minter_v1.0.1.sol	Minter_v1.0.1.sol
SGT	SGT.sol	SGT.sol
STK	SmartTimelock.sol	SmartTimelock.sol
SVG	SmartVesting.sol	SmartVesting.sol
STA	stakingPools.sol	stakingPools.sol
VET	vEth2.sol	vEth2.sol

File Dependency Graph




Finding Summary
5%
37%
58%
Medium
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
MIN-01M	Ambiguous Setter Function	Volatile Code	Medium	✓
MIN-02M	Inexistant Input Sanitization	Volatile Code	Minor	✓
MIN-03M	Inexistant Input Sanitization	Volatile Code	Minor	✓
MIN-04M	Typo in the Error Message	Coding Style	Informational	✓

Static Analysis Findings

ID	Title	Type	Severity	Resolved
MIN-01S	Usage of `transfer()` for sending Ether	Volatile Code	Minor	✓
MIN-02S	Potential Re-Entrancy	Volatile Code	Minor	✓
MIN-03S	Potential Re-Entrancy	Volatile Code	Minor	✓
MIN-04S	Unlocked Compiler Version	Language Specific	Informational	✓
MIN-05S	State Layout	Gas Optimization	Informational	✓
MIN-06S	Visibility Specifiers Missing	Language Specific	Informational	✓
MIN-07S	Redundant Variable Initialization	Coding Style	Informational	✓
MIN-08S	Redundant Type Cast	Gas Optimization	Informational	✓
MIN-09S	Non-Restricting Conditional	Volatile Code	Informational	✓
MIN-10S	Boolean Comparison	Gas Optimization	Informational	✓
MIN-11S	Change to `constant` Variable	Gas Optimization	Informational	✓
STK-01S	Mutability Optimization	Gas Optimization	Informational	✓
SVG-01S	Mutability Optimization	Gas Optimization	Informational	✓
STA-01S	Potential Re-	Volatile Code	Minor	✓

	Entrancy			
STA-02S	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	✓

MIN-01M: Ambiguous Setter Function

Type	Severity	Location
Volatile Code	Medium	Minter_v1.0.1.sol L559-L561

Description:

The `donate()` function directly updates the state of the contract, namely the `curValidatorShares` state variable, yet it publicly accesssible and does not restrict the input values.

Recommendation:

We advise to revise the linked function.

Alleviation:

The development team acknowledged this exhibit and decided to omit any contract state update from the `donate()` function, apart from the receiving Ether.

MIN-02M: Inexistant Input Sanitization

Type	Severity	Location
Volatile Code	Minor	Minter_v1.0.1.sol L587-L589

Description:

Although the access is restricted to anyone but the owner, the `setNumValidators()` can set the number of validators to zero.

Recommendation:

We advise to restrict the input values, accepting non-zero values only.

Alleviation:

The development team opted to consider our references and added a `require` statement ensuring that there will be at least one validator to the system.

MIN-03M: Inexistant Input Sanitization

Type	Severity	Location
Volatile Code	Minor	Minter_v1.0.1.sol L606-L618

Description:

Although the access is restricted to anyone but the owner, the `setMinter()` function fails to check the value of the input address.

Recommendation:

We advise to add a `require` statement, checking the input against the zero address.

Alleviation:

The development team opted to consider our references and added a `require` statement ensuring that the new minter address will be different than the zero address.

MIN-04M: Typo in the Error Message

Type	Severity	Location
Coding Style	Informational	Minter_v1.0.1.sol L543

Description:

The linked error message string contains a typo.

Recommendation:

We advise to update the linked message string.

Alleviation:

The development team opted to consider our references and fixed the typo in the linked error message.

MIN-01S: Usage of `transfer()` for sending Ether

Type	Severity	Location
Volatile Code	Minor	Minter_v1.0.1.sol L554, L638

Description:

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300`. This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation:

We advise that the linked `.transfer()` and `.send()` calls are substituted with the utilization of the `sendValue()` function from the `Address.sol` implementation of OpenZeppelin either by directly importing the library or copying the linked code.

Alleviation:

The development team opted to consider our references and utilized the `sendValue()` function from the `Address.sol` library for the linked statements.

MIN-02S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	Minor	Minter_v1.0.1.sol L566-L585

Description:

The `depositToEth2()` function updates the state of the contract after an external call.

Recommendation:

We advise to move the statement in L584 before the external call (L578-L583).

Alleviation:

The development team opted to consider our references and moved the external call at the end of the function.

MIN-03S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	Minor	Minter_v1.0.1.sol L625-L640

Description:

The `withdrawAdminFee()` function updates the state of the contract after an external call.

Recommendation:

We advise to move the statement in L639 before the external call (L638).

Alleviation:

The development team opted to consider our references and moved the external call at the end of the function.

MIN-04S: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Minter_v1.0.1.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team opted to consider our references and locked the compiler to version `0.7.5`.

MIN-05S: State Layout

Type	Severity	Location
Gas Optimization	Informational	Minter_v1.0.1.sol L424

Description:

The state of the contract is not tightly packed in 256-bit slots.

Recommendation:

We advise to move the `disableWithdrawRefund` state variable adjacent to the `BETHTokenAddress` one, striving for a tight 256-bit packing.

Alleviation:

The development team opted to consider our references and moved the `disableWithdrawRefund` state variable before the `BETHTokenAddress` one.

MIN-06S: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	Minter_v1.0.1.sol L416, L429

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

The development team opted to consider our references and added explicit visibility specifiers for the linked variables.

MIN-07S: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	Minter_v1.0.1.sol L444, L445, L446, L447

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint / int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

The development team opted to consider our references and removed the redundant code.

MIN-08S: Redundant Type Cast

Type	Severity	Location
Gas Optimization	Informational	Minter_v1.0.1.sol L504

Description:

The `msg.value` global variable is already of `uint256` data type.

Recommendation:

We advise to remove the redundant type casting.

Alleviation:

The development team opted to consider our references and removed the redundant type casting.

MIN-09S: Non-Restricting Conditional

Type	Severity	Location
Volatile Code	Informational	Minter_v1.0.1.sol L537-L540, L626-L629

Description:

The linked `require` statements do not restrict the subsequent functionality, as the conditionals will always yield `true`.

Recommendation:

We advise to revise the linked conditionals.

Alleviation:

The development team opted to consider our references and removed the redundant code.

MIN-10S: Boolean Comparison

Type	Severity	Location
Gas Optimization	Informational	Minter_v1.0.1.sol L527

Description:

The linked `if` conditional redundantly compares two boolean variables.

Recommendation:

We advise to directly utilize the value of the `disableWithdrawRefund` state variable instead.

Alleviation:

The development team opted to consider our references and directly used the boolean value stored in the `disableWithdrawRefund` state variable instead.

MIN-11S: Change to `constant` Variable

Type	Severity	Location
Gas Optimization	Informational	Minter_v1.0.1.sol L413

Description:

The `mainnetDepositContractAddress` state variable is never updated after its declaration.

Recommendation:

We advise to change the mutability of the linked state variable to `constant`.

Alleviation:

The development team opted to consider our references and changed the mutability of the linked state variable to `constant`.

STK-01S: Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	SmartTimelock.sol L90

Description:

This contract deviates from Badger's smart timelock contract by not following the `initializable` pattern. Hence, the linked state variable mutability can be optimized.

Recommendation:

We advise to change the mutability specifier of the linked state variable to `immutable`.

Alleviation:

The development team opted to consider our references and changed the mutability specifier of the linked state variable to `immutable`.

SVG-01S: Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	SmartVesting.sol L182

Description:

This contract deviates from Badger's smart vesting contract by not following the `initializable` pattern. Hence, the linked state variable mutability can be optimized.

Recommendation:

We advise to change the mutability specifier of the linked state variable to `immutable`.

Alleviation:

The development team opted to consider our references and changed the mutability specifier of the linked state variable to `immutable`.

STA-01S: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	Minor	stakingPools.sol L675

Description:

The linked code segment updates the state of the contract after an external call.

Recommendation:

We advise to execute the external call at the end of the function, hence following the Checks-Effects-Interactions pattern.

Alleviation:

The development team opted to consider our references and moved the external call after the contract's state update.

STA-02S: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	stakingPools.sol L884-L887

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

The development team opted to consider our references and used the `safeTransfer()` function from the `SafeERC20.sol` library for the linked statement, also removing the `require` statement as the said function does not return a value.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.