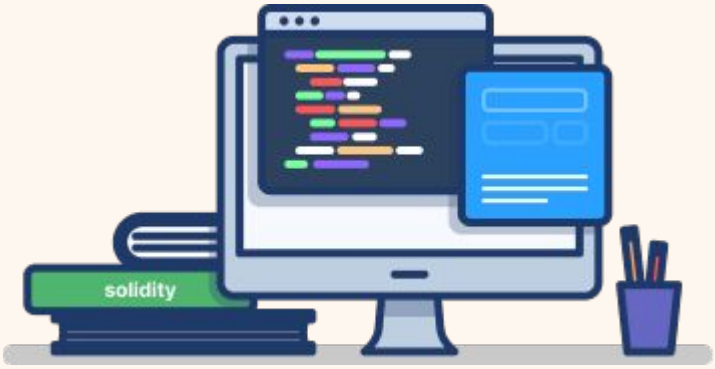
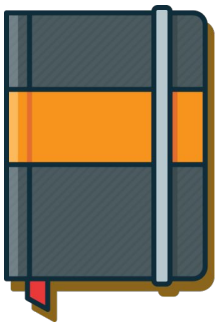
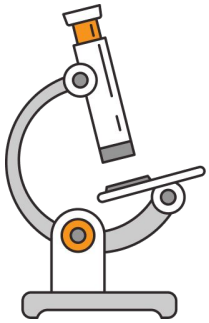
Coinbae Audit

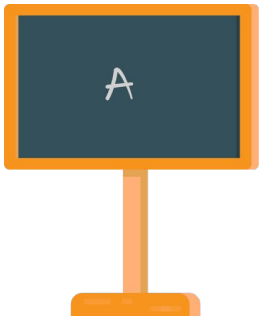


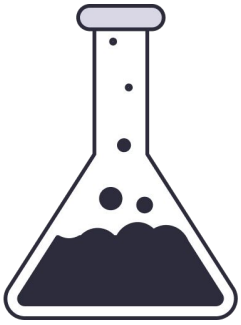
Vault from Deus Finance January 2021

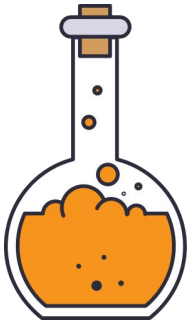
Contents











Introduction

Audit:

In January 2021 Coinbae's audit report division performed an audit for the Deus Finance team (Vault Contract).

https://github.com/deusfinance/vault-backend/blob/main/contracts/Vault.sol

Deus Finance:

DEUS lets you trade real-world assets and derivatives, like stocks and commodities, directly on the Ethereum blockchain.

As described in the Deus Finance litepaper.

DEUS finance is a Decentralized Finance (DeFi) protocol that allows bringing any verifiable digital and non-digital asset onto the blockchain. It boosts the transfer of value across many different markets and exchanges with unprecedented ease, transparency, and security. The launch system is currently being built on the Ethereum-blockchain and will be chain-agnostic in the future. It started out originally as development on a tool to manage the asset basket for a community crypto investment pool. This turned into the vision of DEUS as a DAO-governed, decentralized platform that holds and mirrors assets.

More information can be found at https://deus.finance/home/.

Introduction

Overview:

Information:
Name: Vault
Type: DAO (Governance)

Pool, Asset or Contract address:
https://github.com/deusfinance/vault-backend/blob/main/contracts/Vault.sol

Supply:
Current: NA

Explorers:
Not deployed at the time of audit.

Websites:
https://deus.finance/home/

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

19	19	0
Identified	Confirmed	Critical

0	10	9
High	Medium	Low

Analysis:
https://github.com/deusfinance/vault-backend/blob/main/contracts/Vault.sol

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 calculateSand(address from, uint256 amount) public view returns (uint256){ [#17]
2. function calculateWithdrawAmount(address to, uint256 amount) public view returns (uint256){ [#20]
3. function calculateTime(address from, uint256 amount) public view returns (uint256){ [#28]
4. function setSandController(address _sandController) public onlyOwner{ [#92]
5. function setTimeController(address _timeController) public onlyOwner{ [#96]
6. function setBlocks(uint256 _startBlock, uint256 _endLockBlock) public onlyOwner{ [#100]
7. function lock(uint256 amount) public { [#137]
8. function withdraw(uint256 amount) public { [#146]
9. function withdrawAllLockedTokens(address to) public onlyOwner { [#175]
10. function withdrawLockedTokens(address to, uint256 amount) public onlyOwner { [#182]

Low Severity: Function state can be restricted to pure

If you can make your function pure, you should always make it pure.

Advantages:

If you make your function pure or view, you can call it for example through web3.js without needing a transaction, without any gas cost and without confirmation delay. Currently, it will not reduce gas cost when executing on the EVM. However, in the future it may help the Solidity compiler to optimize your program. It will prevent you from accidentally reading or writing contract state in functions where you don't want to.

Disadvantage:
None

Affected lines:

1. function calculateSand(address from, uint256 amount) public view returns (uint256){ [#17]
   ^ (Relevant source part starts here and spans across multiple lines).
2. function calculateWithdrawAmount(address to, uint256 amount) public view returns (uint256){ [#20]
   ^ (Relevant source part starts here and spans across multiple lines).
3. function calculateTime(address from, uint256 amount) public view returns (uint256){ [#28]
   ^ (Relevant source part starts here and spans across multiple lines).

Low severity: Solidity style guide naming convention issues found

Description:

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.

Affected lines:

1. Parameter Vault.setSandController(address)._sandController (../../../data/Vault.sol#92) is not in mixedCase
2. Parameter Vault.setTimeController(address)._timeController (../../../data/Vault.sol#96) is not in mixedCase
3. Parameter Vault.setBlocks(uint256,uint256)._startBlock (../../../data/Vault.sol#100) is not in mixedCase
4. Parameter Vault.setBlocks(uint256,uint256)._endLockBlock (../../../data/Vault.sol#100) is not in mixedCase
5. Parameter Vault.sandAndTimeAmount(uint256,address)._user (../../../data/Vault.sol#116) is not in mixedCase
6. Parameter Vault.lockFor(uint256,address)._user (../../../data/Vault.sol#122) is not in mixedCase

Advice:

Make sure the contract complies with the https://docs.soliditylang.org/en/v0.8.0/style-guide.html

Informative: Overpowered user

Description:

Functions on the Vault.sol contract are callable only from one address. Because this is the DAO address the risk is lowered from HIGH to INFO.

Line 92:

```
function setSandController(address _sandController) public onlyOwner{
        sandController = SandController(_sandController);
    }
```

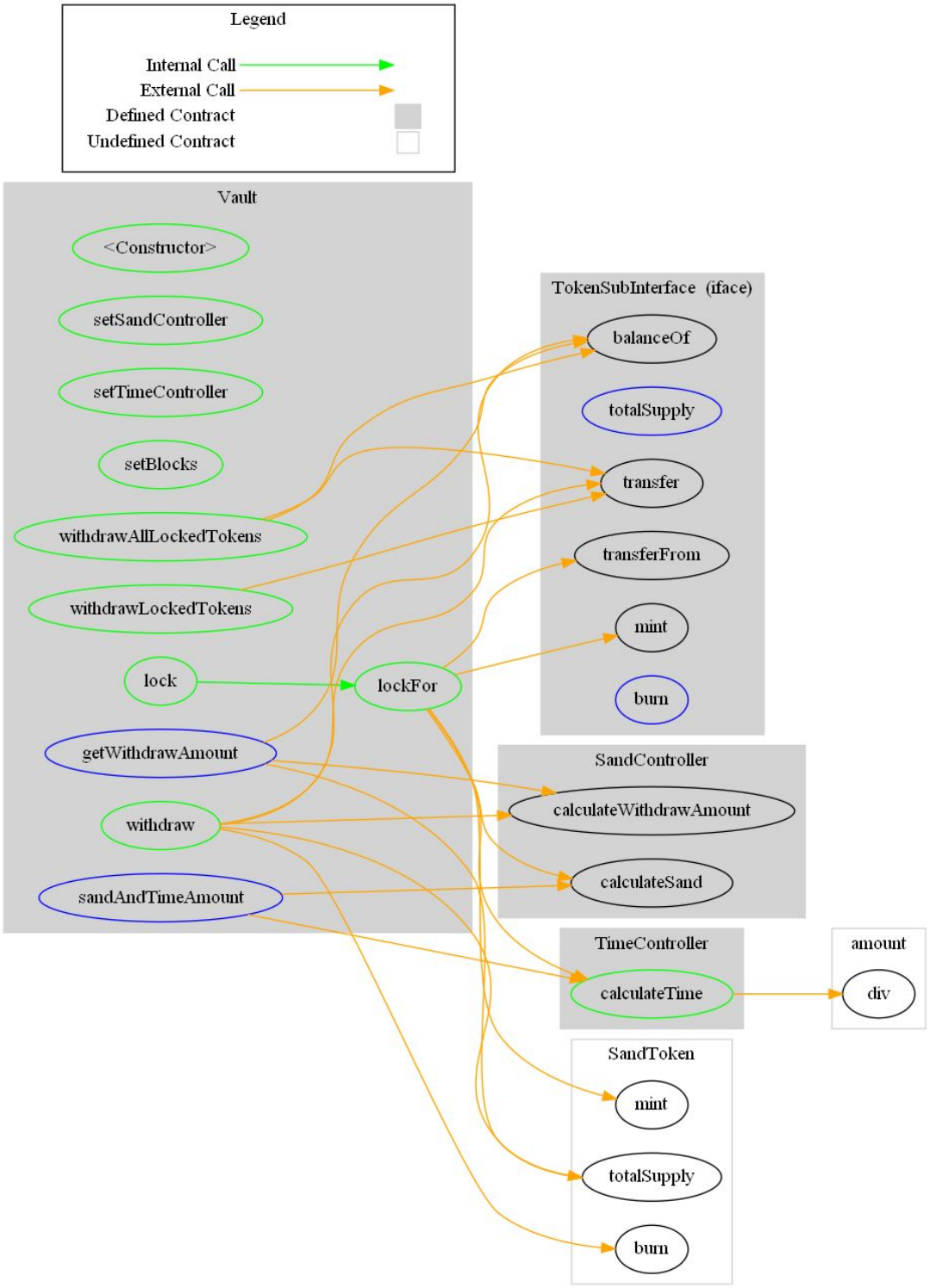
Line 96:

```
function setTimeController(address _timeController) public onlyOwner{
        timeController = TimeController(_timeController);
    }
```

Line 100:

```
function setBlocks(uint256 _startBlock, uint256 _endLockBlock) public
onlyOwner{
        startBlock = _startBlock;
        endLockBlock = _endLockBlock;
    }
```

Contract Flow



Audit Team

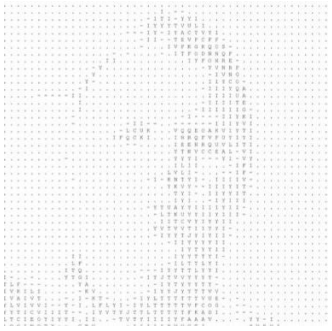
Team Lead: Eelko Neven

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 19 findings, 10 medium and 9 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 Deus Finance platform. 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 Deus Finance Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.