BLOCKSTARS TECHNOLOGY

it@blockstars.com.au

SMART CONTRACTS CODE REVIEW AND SECURITY ANALYSIS REPORT

Handle.Fi

22/07/2021

Table of Contents

INTRODUCTION

This report contains confidential information of audit summary of smart contract programs running in Handle.fi project. It analyses security vulnerabilities, smart contract best practices, and possible attacks, using popular analysis tools and linters. We outlined our systematic approach to evaluate potential security issues in core smart contracts in Handle.fi, and provide additional suggestions for improvement.

The basic information of Handle.fi

Item	Description
Issuer	Handle.fi
Website	https://handle.fi/
Source	Smart contract programs
Language	Solidity
Blockchain	Ethereum
Git Repository	https://github.com/handle-fi/handle-vue
Audit method	Static analysis using symbolic execution

PROJECT SCOPE AND SPECIFICATIONS

Scope of this project is to identify smart contract vulnerabilities to improve the coding practice in smart contract programs implemented in Handle.fi project. We classified the security vulnerabilities of smart contracts in three categories according to their impact level, such as critical, medium, and low class of vulnerabilities. Moreover, we used the impact level definitions from the research of Chen et al. and assigned proper impact values for the identified smart contract vulnerabilities in Handle.fi project.

IP1: It raises critical behaviours and attackers can make benefit by using this vulnerability.

IP2: It raises critical behaviours and attackers cannot make benefit using this vulnerability.

IP3: It raises critical behaviours and attackers cannot trigger them externally (If they trigger, they cannot make benefit.

IP4: Contract works normally, and it leads to potential risks of errors when external programs call the contract.

IP5: It works normally, and it will not lead risks for external callers. But there is no re-usability, and it leads to gas wastage.

Impact Levels	Severity
IP1	High Critical
IP2	Moderate Critical
IP3	Average Critical
IP4	Less Critical
IP5	Normal Risk

ANALYSIS RESULT AND SUGGESTIONS

We used different tools and litters to analyse given smart contracts from Handle.fi project. The major tools we used to analyse smart contracts are Mythx, Osiris, and Solhint linter. The platforms we integrated to test the contracts are Remix, Visual Studio Code, Truffle, and Openzepplin Test Environment.

Smart Contract: Comptroller.sol

FINDING 1

Issue: Dependence on predictable environment variable

Severity: **IP4 (Less Critical)**

Contract: Comptroller

Function name:

mintWithoutCollateral(uint256,address,uint256) - Line: 195

dueBy(deadline) - Line: 199

mintWithEth(uint256,address,uint256) – Line: 72

dueBy(deadline) – Line: 80

burn(uint256,address,uint256) – Line: 224

dueBy(deadline) – Line: 228

mint(uint256,address,address,uint256,uint256)

dueBy(deadline) – Line – 102

Description:

- A control flow decision is made based on The block.timestamp environment variable. The block.timestamp environment variable is used to determine a control flow decision.

- Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks.

- It is recommended not to use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Exact place of usage:

In file: /contracts /interfaces/IValidator.sol:6

require(block.timestamp <= date, "Transaction has exceeded deadline")

Smart Contract Weakness Classification: SWC-116

FINDING 2

Issue: Multiple calls are executed in the same transaction

Severity: **IP3 (Average Critical)**

Interface: IInterest.sol

Function name:

getCurrentR() – Line: 8

Description:

- This call is executed following another call within the same transaction.
- It is possible that the call never gets executed if a prior call fails permanently.
- This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e, they're part of your own codebase).

Exact place of usage:

In file: /contracts /interfaces/IInterest.sol:8

Smart Contract Weakness Classification: SWC-113

Smart Contract: Handle.sol

FINDING 1

Issue: State access after external call

Severity: **IP4 (Less Critical)**

Function name:

setCollateralToken(address,uint256,uint256,uint256) - Line: 145

Description:

- The contract account state is accessed after an external call to a fixed address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted.
- Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Exact place of usage:

In file: /tmp/Bank/Handle.sol:157

```
collateralDetails[_token] = CollateralData({
        mintCR: _mintCR,
        liquidationFee: _liquidationFee,
        interestRate: _interestRatePerMille
    })
```

Smart Contract Weakness Classification: SWC-107

Smart Contract: Treasury.sol

FINDING 1

Issue: The caller can redirect execution to arbitrary bytecode locations.

Severity: **IP2 (Moderate Critical)**

Contract: ReentrancyGuardUpgradeable.sol

Function name: nonReentrant(depositCollateral(…..) – line 106

Description:

- It is possible to redirect the control flow to arbitrary locations in the code. This may allow an attacker to bypass security controls or manipulate the business logic of the smart contract.
- Avoid using low-level-operations and assembly to prevent this issue.

Exact place of usage:

external override nonReentrant {

_depositCollateral(msg.sender, to, amount, collateralToken, fxToken);

} – line: 105-107

Smart Contract Weakness Classification: SWC-127

Smart Contract: VaultLibrary.sol

FINDING 1

Issue: Multiple Calls are executed in a Single Transaction

Severity: **IP4 (Less Critical)**

Contract: VaultLibrary

Function name:

getCurrentRatio(address,address)

Description:

- This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently.
- This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).
- Avoid using low-level-operations and assembly to prevent this issue.

Exact place of usage:

In file: /tmp/Bank/VaultLibrary.sol:112

.div(1 ether);

}

Smart Contract Weakness Classification: SWC-113

FINDING 2

Issue: Integer Arithmetic Bug

Severity: **IP2 (Moderate Critical)**

Contract: VaultLibrary

Function name:

getCurrentRatio(address,address)

Description:

- The arithmetic operator can overflow.
- It is possible to cause an integer overflow or underflow in the arithmetic operation.

Exact place of usage:

In file: /tmp/Bank/VaultLibrary.sol:112

.div(1 ether);

}

Smart Contract Weakness Classification: SWC-101

Smart Contract: Interest.sol

FINDING 1

Issue: Dependence on predictable environment variable

Severity: **IP4 (Less Critical)**

Contract: Interest

Function name: charge()

Description:

- A control flow decision is made based on The block.timestamp environment variable. The block.timestamp environment variable is used to determine a control flow decision.

- Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Exact place of usage:

In file: /tmp/Bank/Interest.sol:57

if (lastChargedDate == block.timestamp) return;

Smart Contract Weakness Classification: SWC-116

Smart Contract: DigitalAssetOracle.sol

No issues found in this contract.

Smart Contract: fxToken.sol

No major issues found. There are compilation warnings detected, that are negligible.

E.g.: Warning: Contract fxToken.sol:8:43 is not start with CapWords

Smart Contract: ChainlinkFiatOracle.sol

No major issues found. There are compilation warnings detected, that are negligible.

Smart Contract: PCT.sol

No major issues found. There are compilation warnings detected, that are negligible.

E.g.: Warning: This declaration has the same name as another declaration. --> Handle/Bank/PCT.sol:84:9: | 84 | uint256 stake = balanceOfStake(account, fxToken, collateralToken); |

Other declaration is here: Handle/Bank/PCT.sol:65:5: | 65 | function stake(|

Smart Contract: PCTCompoundInterface.sol

No issues found in this contract.

Smart Contract: fxKeeperPool.sol

No major issues found. There are compilation warnings detected, that are negligible.

E.g.: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning. --> Handle/Bank/fxKeeperPool.sol:340:9: | 340 | uint256[] memory collateralAmounts, |

Smart Contract: Liquidator.sol

FINDING 1

Issue: Dependence on predictable environment variable

Severity: **IP4 (Less Critical)**

Function name: buyCollateralFromManyVaults(uint256,address,address[],uint256) – Line no: 98

dueBy(deadline) – Line no: 106

Description:

- A control flow decision is made based on The block.timestamp environment variable. The block.timestamp environment variable is used to determine a control flow decision.
- It is recommended not to use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Exact place of usage:

In file: /contracts /interfaces/IValidator.sol:6

require(block.timestamp <= date, "Transaction has exceeded deadline")

Smart Contract Weakness Classification: **SWC ID: 116**

EXECUTIVE SUMMARY

Contracts	Findings & SWC	Status
Comptroller.sol	Dependence on predictable environment variable - SWC-116	**IP4 (Less Critical)**
	Multiple calls are executed in the same transaction – SWC-113	**IP3 (Average Critical)**
Handle.sol	State access after external call – SWC-107	**IP4 (Less Critical)**
Treasury.sol	The caller can redirect execution to arbitrary bytecode locations. – SWC-127	**IP2 (Moderate Critical)**
VaultLibrary.sol	Multiple Calls are executed in a Single Transaction - SWC-113	**IP3 (Average Critical)**
	Integer Arithmetic Bug - SWC-101	**IP2 (Moderate Critical)**
Interest.sol	Dependence on predictable environment variable - SWC-116	**IP4 (Less Critical)**
DigitalAssetOracle.sol	**N/A**	**N/A**
fxToken.sol	**N/A**	Compilation warnings
ChainLinkFiatOracle.sol	**N/A**	Compilation warnings
PCT.sol	**N/A**	Compilation warnings
IPCTProtocolInterface.sol	**N/A**	**N/A**
fxKeeperPool.sol	**N/A**	Compilation warnings
Liquidator.sol	Dependence on predictable environment variable - SWC-116	**IP4 (Less Critical)**