$BOARD
Smart Contract Audit

PRITOM RAJKHOWA

Soteria

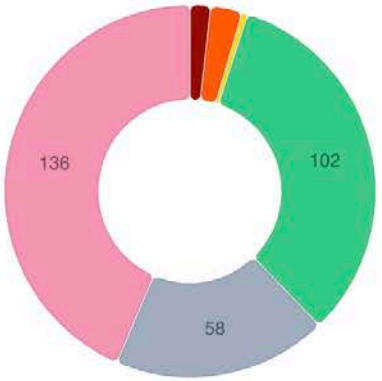
Audit Summary

Contract Address	0xD8513c22Dd61161ba3872859A6D10eB1612Df742

Project Name	MAX_CONTRACT
Contract Type	Smart Contract
Language	Solidity
Codebase	File Scan
Audit Methodology	Static Scanning

Soteria

Findings Summary



Critical	5	Low	102
High	8	Informational	58
Medium	1	Gas	136

Soteria

Critical

```
678         */
679         function burn(uint256 amount) public virtual {
680             _burn(_msgSender(), amount);
681         }
```

PUBLIC BURN

The contract was found to be using public or an external burn function. The function was missing access control to prevent another user from burning their tokens. Also, the burn function was found to be using a different address than msg.sender.

Soteria

Critical

```
694     function burnFrom(address account, uint256 amount) public virtual {
695         uint256 decreasedAllowance = allowance(account, _msgSender()).sub(
696             amount,
697             "ERC20: burn amount exceeds allowance"
698         );
699
700         _approve(account, _msgSender(), decreasedAllowance);
701         _burn(account, amount);
702     }
```

PUBLIC BURN

The contract was found to be using public or an external burn function. The function was missing access control to prevent another user from burning their tokens. Also, the burn function was found to be using a different address than msg.sender.

Soteria

Critical

```
216     /**
217      * @dev Function for withdraw staked token.
218      * Use to withdraw user all remaining staked token and delete user info
219      */
220     function withdrawToken() external nonReentrant afterStakeEnded {
221         require(
222             userInfo[msg.sender].withdrawnAmount == 0,
223             "[StakingV2.withdrawToken] user already withdrawn"
224         );
225         require(
226             getUserUnclaimAmount(msg.sender) == 0,
227             "[StakingV2.withdrawToken] unclaim amount should be zero before
228         );
229         uint256 userStakeAmount = userInfo[msg.sender].stakeAmount;
230         userInfo[msg.sender].withdrawnAmount += userStakeAmount;
231         stakingFactory.vaultTransferTokenToAddress(
232             msg.sender,
233             tokenAddress,
234             userStakeAmount
235         );
236
237         emit TokenWithdrawn(msg.sender, userStakeAmount);
238     }
```

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

Soteria

Critical

```
240     /**
241      * @dev Function for claim staking reward.
242      * Use to transfer reward to user wallet and increase user claim amoun
243      */
244     function claimReward() external nonReentrant afterStakeStarted {
245         uint256 unclaimAmount = getUserUnclaimAmount(msg.sender);
246         require(
247             unclaimAmount > 0,
248             "[StakingV2.claimReward] No claimable reward"
249         );
250         userInfo[msg.sender].claimedAmount += unclaimAmount;
251         distributedReward += unclaimAmount;
252         stakingFactory.vaultTransferTokenToAddress(
253             msg.sender,
254             tokenAddress,
255             unclaimAmount
256         );
257
258         emit RewardClaimed(msg.sender, unclaimAmount);
259     }
260
```

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

Soteria

Critical

```
192     function stakeToken(uint256 _tokenAmount)
193         external
194         whenNotPaused
195         beforeStakeStarted
196     {
197         require(
198             _tokenAmount + userInfo[msg.sender].stakeAmount <= userStakeLi
199             "[StakingV2.stakeToken] total stake amount should be less than
200         );
201         require(
202             _tokenAmount + poolStakeTotal <= poolStakeLimit,
203             "[StakingV2.stakeToken] total stake amount should be less than
204         );
205         stakingFactory.vaultPayWithToken(
206             msg.sender,
207             tokenAddress,
208             _tokenAmount
209         );
210         userInfo[msg.sender].stakeAmount += _tokenAmount;
211         poolStakeTotal += _tokenAmount;
212
213         emit TokenStaked(msg.sender, _tokenAmount);
214     }
215
```

INCORRECT ACCESS CONTROL

Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract.

High

```
108             currentTotalToken = _totalToken;
109             IERC20(_tokenAddress).transferFrom(msg.sender, address(this), _tota
110         }
```

UNCHECKED TRANSFER

Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the transfer or transferFrom function.

Soteria

High

```
99          endDate = _endDate;
100          for (uint256 i = 0; i < _roundPaid.length; i++) {
101              roundPaid.push(_roundPaid[i]);
102
103              emit RoundPaidAdded(i, _roundPaid[i].paidIntervals, _roundPaid[
104          }
105          grantRole(PROJECTOWNER, _projectOwner);
106          uint256 _totalToken = _totalBuyingLimit * _rate / TOKENDECIMAL;
107          totalToken =  _totalToken;
108          currentTotalToken = _totalToken;
109          IERC20(_tokenAddress).transferFrom(msg.sender, address(this), _tota
```

UNCHECKED ARRAY LENGTH

Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of Gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed.

for (uint256 i = 0; i < array.length ; i++) { costlyFunc(); }

This becomes a security issue if an external actor influences array.length.

E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.

Soteria

High

```
432         *
433         * - `spender` cannot be the zero address.
434         */
435        function approve(address spender, uint256 amount)
436            public
437            virtual
438            override
439            returns (bool)
440        {
441            _approve(_msgSender(), spender, amount);
442            return true;
443        }
444
```

APPROVE FRONT-RUNNING ATTACK

The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.
This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account.
Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function.
The function approve can be front-run by abusing the _approve function.

Soteria

High

```
458     function transferFrom(
459         address sender,
460         address recipient,
461         uint256 amount
462     ) public virtual override returns (bool) {
463         _transfer(sender, recipient, amount);
464         _approve(
465             sender,
466             _msgSender(),
467             _allowances[sender][_msgSender()].sub(
468                 amount,
469                 "ERC20: transfer amount exceeds allowance"
470             )
471         );
472         return true;
473     }
474
```

APPROVE FRONT-RUNNING ATTACK

The transferFrom() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.

This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account.

Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function.

High

```
693         */
694         function burnFrom(address account, uint256 amount) public virtual {
695             uint256 decreasedAllowance = allowance(account, _msgSender()).sub(
696                 amount,
697                 "ERC20: burn amount exceeds allowance"
698             );
699 
700             _approve(account, _msgSender(), decreasedAllowance);
701             _burn(account, amount);
702         }
703     }
```

APPROVE FRONT-RUNNING ATTACK

The burnFrom() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.
This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account.
Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function.
The function burnFrom can be front-run by abusing the _approve function.

Soteria