SALUS SECURITY

NOV 2023

CODE SECURITY ASSESSMENT

ANOTHER WORLD

Overview

Project Summary

- Name: Another World
- Platform: Klaytn
- Language: Solidity
- Repository:
    - https://github.com/WMB-Another-World/aw-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Another World
Version	v2
Type	Solidity
Dates	Nov 29 2023
Logs	Nov 24 2023; Nov 29 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	4
Total informational issues	5
Total	9

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Duplicate stakingTokens will result in pool.accRewardPerShare being updated incorrectly	Low	Business Logic	Acknowledged
2	The rewardToken mismatch between ERC20StakingManager and ERC20StakingRewardFunds could affect users' harvest	Low	Business Logic	Resolved
3	Implementation contract could be initialized by everyone	Low	Business Logic	Resolved
4	Centralization risk	Low	Centralization	Acknowledged
5	Weak condition checks	Informational	Data Validation	Acknowledged
6	Function parameter overwrite is only used in the event	Informational	Code Quality	Acknowledged
7	Use of the magic number	Informational	Code Quality	Acknowledged
8	Typos	Informational	Code Quality	Acknowledged
9	Gas optimization suggestions	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Duplicate stakingTokens will result in pool.accRewardPerShare being updated incorrectly

Severity: Low	Category: Business Logic

Target:
- contracts/dex/ERC20StakingManager.sol

Description

During the update of the pool's accRewardPerShare, the tokenSupply is the specified stakingToken balance of the ERC20StakingManager contract. If there are two pools with the same stakingToken, the tokenSupply is wrong, causing the pool's accRewardPerShare being updated incorrectly.

contracts/dex/ERC20StakingManager.sol:L114-L124

```
uint256 tokenSupply = stakingTokens[pid].balanceOf(address(this));

if (tokenSupply > 0) {
        ...
        pool.accRewardPerShare =
                pool.accRewardPerShare +
                (((_reward * ACC_REWARD_PRECISION) / tokenSupply));
        }
}
```

However, there is no duplicate checking when adding a pool.

contracts/dex/ERC20StakingManager.sol:L79-L92

```
function add(uint256 allocPoint, IERC20 stakingToken) public onlyAnotherWorldAdmin {
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint + allocPoint;
        stakingTokens.push(stakingToken);

        poolInfo.push(
            PoolInfo({
                allocPoint: allocPoint,
                lastRewardBlock: lastRewardBlock,
                accRewardPerShare: 0
            })
        );
        emit DexStakingPoolAdded(stakingTokens.length - 1, allocPoint, stakingToken);
}
```

Recommendation

It is recommended to ensure that there are no duplicate staking tokens before adding.

Status

This issue has been acknowledged by the team.

2. The rewardToken mismatch between ERC20StakingManager and ERC20StakingRewardFunds could affect users' harvest

Severity: Low	Category: Business Logic
Target: - contracts/dex/ERC20StakingManager.sol - contracts/dex/ERC20StakingRewardFunds.sol	

Description

The ERC20StakingManager has the ability to transfer reward tokens, which come from the ERC20StakingRewardFunds contract, to users.

The AnotherWorldAdmin of the ERC20StakingRewardFunds contract can change the address of rewardToken at any time. Thus, the addresses of rewardToken in these two contracts may be different, which could result in insufficient balance when users try to harvest.

Recommendation

Consider removing the setReward() function in the ERC20StakingRewardFunds contract.

Status

This issue has been resolved by the team with commit 8e7d77c.

3. Implementation contract could be initialized by everyone

Severity: Low	Category: Business Logic

Target:
- contracts/distributor/Vault.sol
- contracts/distributor/Distributor.sol
- contracts/dex/ERC20StakingManager.sol
- contracts/dex/ERC20StakingRewardFunds.sol

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the initialize() function in the implementation contracts of Vault, Distributor, ERC20StakingManager, and ERC20StakingRewardFunds.

Recommendation

To prevent the implementation contract from being used, consider invoking the _disableInitializers function in the constructor of the above-mentioned contracts to automatically lock it during the deployment.

Status

This issue has been resolved by the team with commit 8e7d77c.

4. Centralization risk

Severity: Low	Category: Centralization
Target: - contracts/dex/ERC20StakingRewardFunds.sol	

Description

There is a privileged admin role in the ERC20StakingRewardFunds contract. The AnotherWorldAdmin of the ERC20StakingRewardFunds contract can withdraw all the tokens.

contracts/dex/ERC20StakingRewardFunds.sol:L34-L37

```
function withdraw() external onlyAnotherWorldAdmin {
        uint256 amount = rewardToken.balanceOf(address(this));
        rewardToken.transfer(msg.sender, amount);
}
```

Should AnotherWorldAdmin's private key be compromised, an attacker could change the address of rewardToken and withdraw all the tokens in the contract.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. Weak condition checks

Severity: Informational	Category: Data Validation
Target: - contracts/distributor/Vault.sol - contracts/distributor/Distributor.sol	

Description

The state variable totalAmount, which is defined in the Vault contract and the Distributor contract, tracks the total amount of tokens in the contract. It increases in the supply() function and decreases when users claim rewards.

In the initialDistribute() function, the parameter _amount is checked against the totalAmount to ensure that the distribution does not exceed the token balance.

contracts/distributor/Distributor.sol:L138-L150

```
function initialDistribute(uint256 _amount) public onlyAnotherWorldAdmin {
        ...
        require(totalAmount >= _amount, 'Distributor: lack of token amount in
distributor');
        ...
}
```

In the additinalDistribute() function, which is used to distribute additional amount, there is no similar check. Thus, initialAmount could be greater than totalAmount after additional distribution, which may cause confusion when users claim rewards.

contracts/distributor/Distributor.sol:L153-L161

```
function additinalDistribute(uint256 _amount) public onlyAnotherWorldAdmin {
        address _guestAddress = guestAddress;
        GuestInfo storage guest = guestInfo;

        require(guest.initialAmount != 0, 'Distributor: guest not exist');
        guest.initialAmount += _amount;

        emit DistributorDistribute(_guestAddress, _amount);
}
```

Recommendation

For the Vault contract, since it is multi-user, consider using a state variable to track the total initial amount across all users and compare it to the totalAmount during distribution.

For the Vault contract and the Distributor contract, consider adding a check to ensure that the initial amount is not greater than totalAmount after additional distribution.

Status

This issue has been acknowledged by the team.

6. Function parameter overwrite is only used in the event

Severity: Informational	Category: Code Quality
Target: - contracts/dex/ERC20StakingManager.sol	

Description

The parameter overwrite is only used in the event, which does not provide any information for off-chain tracking.

contracts/dex/ERC20StakingManager.sol:L94-98

```
function set(uint256 _pid, uint256 _allocPoint, bool overwrite) public
onlyAnotherWorldAdmin {
        totalAllocPoint = totalAllocPoint - poolInfo[_pid].allocPoint + _allocPoint;
        poolInfo[_pid].allocPoint = _allocPoint;
        emit DexStakingPoolSet(_pid, _allocPoint, overwrite);
}
```

Recommendation

Consider deleting the parameter overwrite and logging the old allocPoint in the event.

Status

This issue has been acknowledged by the team.

7. Use of the magic number

Severity: Informational	Category: Code Quality
Target: - contracts/distributor/Vault.sol	

Description

Magic numbers refer to the use of hard-coded numerical values in code without any explanation or context. They can make the code difficult to read, understand, and maintain.

In the _computeAvailableAmount function, numerical calculations are based on magic numbers.

contracts/distributor/Vault.sol:L335

```
vestTermPercentage = (vestTerm * 100000000) / (vestEndTime - _lastClaimTime);
```

contracts/distributor/Vault.sol:L344

```
vestTermPercentage = (vestTerm * 100000000) / (vestEndTime - vestStartTime);
```

contracts/distributor/Vault.sol:L351

```
uint256 additionalAvailableAmount = (remainAmount * vestTermPercentage) / 100000000;
```

Recommendation

To improve the code's readability and facilitate refactoring, consider defining a constant for every magic number, giving it a clear and self-explanatory name.

Status

This issue has been acknowledged by the team.

8. Typos

Severity: Informational	Category: Code Quality

Target:
- contracts/distributor/Vault.sol
- contracts/distributor/Distributor.sol

Description

There are typos in the codes below.

contracts/distributor/Distributor.sol:L11

```
uint256 initialAmount; // Amount of tokens guest initialy had
```

contracts/distributor/Vault.sol:L11

```
uint256 initialAmount; // Amount of tokens user initialy had
```

Initialy should be initially.

contracts/distributor/Distributor.sol:L153

```
function additinalDistribute(uint256 _amount) public onlyAnotherWorldAdmin
```

contracts/distributor/Vault.sol:L250

```
function additinalDistribute(address _user, uint256 _amount) public
onlyAnotherWorldAdmin
```

additinalDistribute should be additionalDistribute.

contracts/distributor/Distributor.sol:L216, L219

```
if (currentTime > vestEndTime) {
            // after veset end
            ...
        } else {
            // before veset end
            ...
}
```

veset should be vest.

contracts/distributor/Vault.sol:L124

```
require(lockupAmount >= _amount, 'Vault: exceeed lockup amount');
```

exceeed should be exceed.

Recommendation

Consider fixing the typos.

Status

This issue has been acknowledged by the team.

9. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - contracts/distributor/Vault.sol	

Description

When the contract itself sends tokens, it is recommended to use transfer directly. The approve() and transferFrom() functions are used when the token sender is not the token owner itself, and then the spender will send the token on behalf of the owner.

contracts/distributor/Vault.sol:L81-82

```
function claim() public nonReentrant {
        ...
        token.approve(address(this), newAvailableAmount);
        token.transferFrom(address(this), msg.sender, newAvailableAmount);
        ...
}
```

contracts/distributor/Vault.sol:L128-131

```
function instantClaim(uint256 _amount) public nonReentrant {
        ...
        token.approve(address(this), userAmount);
        token.transferFrom(address(this), msg.sender, userAmount);
        token.approve(address(this), burnAmount);
        token.transferFrom(address(this), burnAddress, burnAmount);
        ...
}
```

contracts/distributor/Vault.sol:L172-175

```
function instantClaimMaxAmount() public nonReentrant {
        ...
        token.approve(address(this), userAmount);
        token.transferFrom(address(this), msg.sender, userAmount);
        token.approve(address(this), burnAmount);
        token.transferFrom(address(this), burnAddress, burnAmount);
        ...
}
```

Recommendation

Consider using the above suggestions to save gas.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 63dd217:

File	SHA-1 hash
contracts/aw20/AW20.sol	5ab19f6b362941cb94c29368d07f3576cf249acd
contracts/distributor/Vault.sol	94b529cbafa4a4196e2d4b91687ea037e7308b43
contracts/distributor/Distributor.sol	7d354932ac9b035404009a7541d93a460aac3b68
contracts/dex/ERC20StakingManager.sol	9189825086059ec4e480faa2bb9577f151125773
contracts/dex/ERC20StakingRewardFunds.sol	830a6aa65b2e173bb8da21cc570ab37ef7fc6381