Woonkly Security Audit (DEX & STAKE)

Heloisa Ceni (https://blog.coinfabrik.com/author/heloisa-ceni/)

April 7, 2021 (https://blog.coinfabrik.com/smart-contract-audit/woonkly-security-audit-dex-stake/)

Contents

Introduction

CoinFabrik was asked to audit the contracts for the Woonkly project. First we will provide a summary of our discoveries and then we will show the details of our findings.

Summary

The contracts audited are from the STAKESmartContractPreRelease (https://github.com/Woonkly/STAKESmartContractPreRelease) (https://github.com/Woonkly/STAKESmartContractPreRelease)repository and DEXsmartcontractsPreRelease (https://github.com/Woonkly/DEXsmartcontractsPreRelease) repository at GitHub. The audit is based on the commit 779522de8afcf9285d660abdcc0fb10a62f6f659 (https://github.com/Woonkly/STAKESmartContractPreRelease/commit/779522de8afcf9285d660abdcc0fb10a62f6f659), and 49777cd67d0227c21437236ab9f42b501a864895 (https://github.com/Woonkly/DEXsmartcontractsPreRelease/commit/49777cd67d0227c21437236ab9f42b501a864895) respectively.

Contracts

The audited contracts are:

● woonklyPOS.sol

● InvestorManager.sol

● WonklyDEX.sol

● StakeManager.sol

● Investing.sol

● Pausabled.sol

● Erc20Manager.sol

Analyses

The following analyses were performed:

● Misuse of the different call methods

● Integer overflow errors

● Division by zero errors

● Outdated version of Solidity compiler

● Front running attacks

● Reentrancy attacks

● Misuse of block timestamps

● Softlock denial of service attacks

● Functions with excessive gas cost

● Missing or misused function qualifiers

● Needlessly complex code and contract interactions

● Poor or nonexistent error handling

● Failure to use a withdrawal pattern

● Insufficient validation of the input parameters

● Incorrect handling of cryptographic signatures

Detailed findings

Severity Classification

Security risks are classified as follows:

● Critical: These are issues that we manage to exploit. They compromise the system seriously. They must be fixed immediately.

● Medium: These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them as soon as possible.

● Minor: These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed when possible.

● Enhancement: These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

Inaccurate coin reserve balance tracking in DEX rewards `uint256 internal _coin_reserve;`

_coin_reserve is an internal variable in the contract WonklyDEX.sol which is used to track the contract's balance between its functions, this variable should reflect `address(this).balance` value. For saving gas, it need not to be updated at every possible chance, but at least after any function uses the transfer function. However in some functions it is not handled well. As is the case of the function `_withdrawReward` which does not update the `_coin_reserve` value in the case that the parameter `isCoin` is set to true (bnb).

As a result, an inaccurate tracking of the contract's balance may occur, although the difference could be small since it originates from rewards, and is immediately corrected at the call of other functions, a malicious group of actors could arrange an attack in a contract and coordinate a massive reward withdrawal at the same time followed by a call to withdrawFunds function in that same attacker's contract.
We conducted three variants of attacks that make use of these flaws:

1. The first attack variant consists in preparing a contract where actors first collect the most rewards possible and then, withdraw them altogether, momentarily producing a significant difference between _coin_reserve variable and the real contract's balance, to be taken advantage by immediately calling the function ***withdrawFunds***, resulting in the returned amount, being higher than it should.
2. Another way to take advantage is using the coinToToken function which also would send more ETH that it should, since it is using `_coin_reserve` to establish the exchange price.
3. Another variant that could be used to exploit this issue is to expect a relatively large ***AddLiquidity*** function call, to the contract and before it executes, send ***WithdrawReward*** call, to lower the real contract balance without lowering the `_coin_reserve` value, which will then add `msg.value to _coin_reserve` variable, that already has a higher value than it should, miscalculating the stake, being less than the victim should receive.

In the function AddLiquidity variable _coin_reserve is set in the following way in line 695:

```
_coin_reserve=_coin_reserve.add(msg.value);
```

Though technically correct, doing `_coin_reserve + msg.value` could drag previously erroneous `_coin_reserve` values. It is always preferable to track it directly by using `address(this).balance`, as is done in other functions of the contract, whenever there is a **bnb** transfer from the contract.

The suggested solution is to add the mentioned ***_coin_reserve*** assignation in the internal function ***_withdrawReward***.

This issue was fixed at commit 51b793cc305d5383a5aa18c85c0d4cadd9930e4a (https://github.com/Woonkly/DEXsmartcontractsPreRelease/commit/51b793cc305d5383a5aa18c85c0d4cadd9930e4a)

Miscalculated decrease value in rewards

In the contract woonklyPOS.sol the function ***_withdrawReward*** is expected to decrease the user rewards of certain token for the claiming amount, but instead the subtraction to the total is done two times (see below lines (1) and (2)), resulting in a miscalculated result:

```
uint256 rew=rewarded( sc, account);
uint256 remainder = rew.sub(amount);                              (1)
if(remainder==0){
_doreward(sc, account, 0);
} else{
require(_decreaseRewards( sc,  account, remainder),"WO:e--");       (2)
```

The intended behavior is to decrease the pending rewards by the amount withdrawn.

When the function ***_withdrawReward*** is called with an amount that is smaller than the total number of the user's rewards, the user receives this amount but this amount is not correctly subtracted from his rewards. Assume a user has a total reward of 100 tokens and the function is called with an amount to claim 95 tokens; then remainder is set by line (1) at remainder=100-95=5. Next, the if evaluates to the second clause and the statement in line (2) executes. After line (2) executes, the user has claimed 95 tokens while `_rewards[sc][account]` has been incorrectly set to 95, when it should have been set 5. The attacker can then call this function again, claiming 90 tokens and receive (90+95=185) which is more than the 100 tokens it was rewarded with.

Two fixes can prevent this issue. First, the if statement may be replaced with

```
_doreward(sc, account, remainder);
```

Which sets it at ***remainder***'s absolute value.

The other option is to replace line (2) by line (2') below calling the internal function ***_decreaseRewards*** but with the ***amount*** parameter's value rather than the final result (since this function already does the subtraction internally and sets the reward's value relative to the previous value, decreasing it by this amount):

```
_decreaseRewards( sc,  account, amount)                  (2’)
```

This issue was fixed at commit bc1f9a3dc4bb138cf6c52104dd93da64f0f78802 (https://github.com/Woonkly/STAKESmartContractPreRelease/commit/bc1f9a3dc4bb138cf6c52104dd93da64f0f78802)

Medium severity

Denial of service in iterations over stakes with multiple accounts

In the contract woonklyPOS.sol, lines 876 and 925 include for iterations that could cause the execution to run out of gas in the case where many stakes happen simultaneously.

Solution: use pagination or a delimiter index parameter that ensures that all stakes get iterated no matter how many there are.

Denial of service in withdraw funds function over iteration

In the contract WonklyDEX.sol in the function at line 246 coin transfer could fail due to the receiver being a contract that rejects payments by not implementing a fallback or a receive function.
In case it fails during the transfer instruction could revert, reverting iteration and the whole transaction.
Solution: Use OpenZeppelin's SafeTransfer function which will ignore the rejected payments instead of reverting.

Minor severity

Reentrancy using custom token

It is possible for an attacker to prepare a malicious ERC20 contract and inject it by using the function ***processReward(address sc, uint256 amount) public*** in the contract WoonklyPOS.sol.

For example, an attacker may develop a contract, with arbitrary ***balanceOf***, ***allowance***, or ***transfer*** functions and then use the address of this contract when calling processReward so that these arbitrary functions are used. In particular, if any of these functions calls ***processReward*** the reentrancy takes effect. This could be combined into an attack depending on this new contract.

This issue was fixed at commit 4c95a4b1b45b99ddb54cba093fe8ed326cf48602 (https://github.com/Woonkly/STAKESmartContractPreRelease/commit/4c95a4b1b45b99ddb54cba093fe8ed326cf48602)

Enhancements

Duplicated require statement

In the WonklyDEX.sol contract, inside the `WithdrawLiquidity` public function, at line 792, there is the following require statement:

```
require( _stakes.StakeExist(_msgSender()),"DX:!");
```

Afterwards, the internal function, `_WithdrawReward` is called with `_msgSender()` as parameter for `account` , and in which it already has that require in line 754

```
 require( _stakes.StakeExist(account),"DX:!")
```

The same happens in the `WithdrawReward` public function, and it's require at line 423, with the same requirement inside the internal function `_withdrawReward` .

We recommend only using that require in the internal functions and removing it from the externals, trusting that it is properly verified internally always.

Enhancement applied at commit 51b793cc305d5383a5aa18c85c0d4cadd9930e4a (https://github.com/Woonkly/DEXsmartcontractsPreRelease/commit/51b793cc305d5383a5aa18c85c0d4cadd9930e4a)

Unnecessary condition

In the WonklyDEX.sol contract at line 465 there is the following if clause which is superfluous, since replacing 0 by ***remainder*** accomplishes the same result. That is, replacing the code

```
if(remainder==0){
            _stakes.changeReward(account,0, isCoin,1);
}else{
            _stakes.changeReward(account,remainder, isCoin,1);
```

with only this line

```
_stakes.changeReward(account,remainder, isCoin,1);
```

Removing the if results in less code and saves some gas.

Enhancement applied at commit 51b793cc305d5383a5aa18c85c0d4cadd9930e4a (https://github.com/Woonkly/DEXsmartcontractsPreRelease/commit/51b793cc305d5383a5aa18c85c0d4cadd9930e4a)

Conclusion

The project is very good written overall and understandable despite the lack of commentaries or documentation. The good use of modifiers makes the code easy to follow and well structured.
We found two critical issues with a major impact that need to be addressed, the “Inaccurate coin reserve balance tracking in DEX rewards” issue and the “Miscalculated decrease value in rewards” issue. These are the most relevant findings.
Also, the “Reentrancy using custom token” issue is worth mentioning. This issue, while appearing low-risk, may open an entry point for more sophisticated attacks that may present a bigger risk in the future.

Related Posts

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-

Katana Smart Contract Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/katana-smart-contract-audit/)

CoinFabrik was asked to audit the contracts for the Katana project. First we will provide…

contracts/katana-
smart-
contract-
audit/)

(https://blog.coinfabrik.com/smart-
contracts/etherparty-
token-
smart-
contract-
security-
audit-
coinfabrik/)

EtherParty Smart Contract Security Audit
(https://blog.coinfabrik.com/smart-contracts/etherparty-token-smart-contract-security-audit-coinfabrik/)

Coinfabrik team has been hired to audit Etherparty smart contracts. Firstly, we will provide a…

(https://blog.coinfabrik.com/smart-
contracts/smart-
contract-
audit-
smart-
contracts/yffii-
smart-
contract-
audit/)

YFFII Smart Contract Audit
(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/yffii-smart-contract-audit/)

CoinFabrik was asked to audit the contracts for the YFFII project. First we will provide…

(https://blog.coinfabrik.com/smart-
contracts/smart-
contract-
audit-
smart-
contracts/bricks4us-
token-
smart-

Bricks4US Token Smart Contract Security Audit
(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/bricks4us-token-smart-contract-security-audit/)

CoinFabrik was asked to audit the contracts for the Bricks4Us token sale. Firstly, we will…

contract-
security-
audit/)

Tags:

audit (https://blog.coinfabrik.com/tag/audit/)

auditoria (https://blog.coinfabrik.com/tag/auditoria/)

auditoria de seguridad (https://blog.coinfabrik.com/tag/auditoria-de-seguridad/)

security audit (https://blog.coinfabrik.com/tag/security-audit/)

smart contract audits (https://blog.coinfabrik.com/tag/smart-contract-audits/)

SHARE ON

(https://www.facebook.com/sharer/sharer.php?u=https://blog.coinfabrik.com/smart-contract-audit/woonkly-security-audit-dex-stake/)
(https://twitter.com/intent/tweet?text=Woonkly%20Security%20Audit%20(DEX%20&%20STAKE)&url=https://blog.coinfabrik.com/smart-contract-audit/woonkly-security-audit-dex-stake/)
(https://www.linkedin.com/shareArticle?mini=true&url=https://blog.coinfabrik.com/smart-contract-audit/woonkly-security-audit-dex-stake/&title=Woonkly%20Security%20Audit%20(DEX%20&%20STAKE)&source=CoinFabrik%20Blog)

← PREVIOUS ARTICLE

The Tension Between Divergence Loss and Profit in Automated Market Makers (https://blog.coinfabrik.com/finance/the-tension-between-divergence-loss-and-profit-in-automated-market-makers/)

NEXT ARTICLE →

BLOX STAKING Audit: Vesting and DEX Contracts (https://blog.coinfabrik.com/uncategorized/blox-staking-audit-vesting-and-dex-contracts/)

You may also like

(https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

Magic Bridge Audit (https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

Smart (https://blog.coinfabrik.com/category/smart-

2 months ago

Smart Contract Audit (https://blog.coinfabrik.com/category/smart-contracts/smart-contract-audit-smart-contracts/)

MintingFactoryV2, BaseUpgradableMarketplace & KODAV3UpgradableGatedMarketplace (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

(https://blog.coinfabrik.com/feed/)

(https://ar.linkedin.com/company/coinfabrik)

(https://twitter.com/coinfabrik)

(https://www.youtube.com/channel/UC2GmjCr7aEz-il31kqOy9aw)

(https://www.facebook.com/CoinFabrik/)

(https://www.reddit.com/r/CoinFabrik/)

(https://github.com/coinfabrik)