SALUS SECURITY

SEP 2024

CODE SECURITY ASSESSMENT

XUNION

Overview

Project Summary

- Name: Xunion - Xdex
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/artixv/xdex/
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Xunion - Xdex
Version	v4
Type	Solidity
Dates	Sep 12 2024
Logs	Aug 26 2024; Sep 02 2024; Sep 11 2024; Sep 12 2024

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	2
Total Low-Severity issues	4
Total informational issues	2
Total	10

Contact

E-mail: support@salusec.io

SALUS

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.

SALUS

Content

SALUS

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.

SALUS

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	RewardContractSetup function has no permission control	High	Access Control	Resolved
2	Everyone could take permission role	High	Access Control	Resolved
3	Initiallpowners's LP may be stolen	Medium	Access Control	Resolved
4	CreatLpVault's approve may fail	Medium	Business Logic	Resolved
5	Use openzeppelin's ecdsa library instead of evm's ecrecover	Low	Cryptography	Resolved
6	Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()	Low	Risky external calls	Resolved
7	Not applicable for fee on transfer token	Low	Business Logic	Resolved
8	Missing events for functions that change critical state	Low	Logging	Resolved
9	Use of floating pragma	Informational	Configuration	Resolved
10	Gas optimization suggestions	Informational	Gas Optimization	Resolved

SALUS

2.2 Notable Findings

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

1. RewardContractSetup function has no permission control

Severity: High	Category: Access Control
Target: - contracts/xunionswappair.sol	

Description

Everyone can set the `rewardContract` by calling the `rewardContractSetup` function.

xunionswappair.sol:L101-L103

```
function rewardContractSetup(address _rewardContract) public {
    rewardContract = _rewardContract;
}
```

In the `_afterTokenTransfer` hook, the `rewardContract` will record the balance of from and to address.

xunionswappair.sol:L141-L148

```
function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override {
    iRewardMini(rewardContract).recordUpdate(from,balanceOf(from));
    iRewardMini(rewardContract).recordUpdate(to,balanceOf(to));
}
```

If the `rewardContract` is a malicious address, it can lead to the following potential issues:

1. The lost records may result in potential loss of rewards for users.
2. Malicious revert could cause the transfer function to become paralyzed.

Recommendation

Add a permission control to the `rewardContractSetup` function.

Status

The team has resolved this issue in commit fc592b8.

SALUS

2. Everyone could take permission role

Severity: High	Category: Access Control

Target:
- contracts/xunionswaplpmanagement.sol

Description

The setPA should check if `msg.sender` and `setPermissionAddress` are equal. However, the function checks if `msg.sender` and `_setPermissionAddress`, which is a parameter, are equal.

xunionswaplpmanagement.sol:L163-L66

```
function setPA(address _setPermissionAddress) external {
    require(msg.sender == _setPermissionAddress, 'X Swap LpManager: Permission
FORBIDDEN');
    newPermissionAddress = _setPermissionAddress;
}
```

The `newPermissionAddress` also can become `setPermissionAddress` by calling the `acceptPA()` function.

xunionswaplpmanagement.sol:L67-L73

```
function acceptPA(bool _TorF) external {
    require(msg.sender == newPermissionAddress, 'X Swap LpManager: Permission
FORBIDDEN');
    if(_TorF){
        setPermissionAddress = newPermissionAddress;
    }
    newPermissionAddress = address(0);
}
```

If the `setPermissionAddress` address loses control, attackers can call the following functions:

1. `settings()` for resetting `factory`, `xVaults` and `lpVault`.
2. `settingMinLpLimit()` for resetting `minLpLimit`.
3. `xLpInfoSettings()` for resetting `xVaults`'s lpSettings.
4. `exceptionTransfer()` for transferring the native token.

Recommendation

Add a permission control to the `setPA()` function.

Status

The team has resolved this issue in commit [fc592b8](fc592b8).

SALUS

3. Initiallpowners's LP may be stolen

Severity: Medium	Category: Access Control
Target: - contracts/xunionswapinterface.sol	

Description

The `xunionswapinterface::xLpSubscribe()` function will record `msg.sender` as the initialLpOwner if LP is not initialized and call xlpmanager's `xLpSubscribe()` function.

xunionswapinterface.sol:L81-L102

```
function xLpSubscribe(address _lp,uint[2] memory _amountEstimated) public
returns(uint[2] memory _amountActual,uint _amountLp) {
    ...
    (TokensAmount,, )= getLpReserve(_lp);
    if(TokensAmount[0]==0){
        initialLpOwner[_lp] = msg.sender;
    }
    ...
    (_amountActual,_amountLp) =
ixLpManager(xlpmanager).xLpSubscribe(_lp,_amountEstimated);
   ...
}
```

The `xunionswaplpmanagement::xLpSubscribe()` function will record `msg.sender` as the initialLpOwner and call `mintXLp()` to mint XLP to lpVault.

xunionswaplpmanagement.sol:L88-L161

```
function xLpSubscribe(address _lp,uint[2] memory _amountEstimated) external lock
returns(uint[2] memory _amountActual,uint _amountLp){
    ...
    if(reserve[0]==0){
        initialLpOwner[_lp] = msg.sender;
        initLpAmount[_lp] = _amountLp;
        ixUnionSwapPair(_lp).mintXLp(lpVault, _amountLp);
    }else{
        ixUnionSwapPair(_lp).mintXLp(msg.sender, _amountLp);
    }
    emit Subscribe(_lp, msg.sender, _amountLp);
}
```

The `xunionswapinterface::initialLpRedeem()` function will call `xlpvaults::initialLpRedeem()` to redeem XLP from lpVault.

xunionswapinterface.sol:L262-L265

```
function initialLpRedeem(address _lp) public returns(uint _amount) {
    _amount = ixLpVaults(xlpvaults).initialLpRedeem(_lp);
    IERC20(_lp).transfer(msg.sender,IERC20(_lp).balanceOf(address(this)));
}
```

The `xunionswaplpvaults::initialLpRedeem()` function will send the LP to the

SALUS

`lpManager::initialLpOwner()` after `lpTimeLimit` time.

xunionswaplpvaults.sol:L262-L265

```
function initialLpRedeem(address _lp) external returns(uint _amount){
    require(lpInitTime[_lp] + lpTimeLimit > block.timestamp,"X SWAP Lp Vaults: Time
Limit");
    require(iLpVaultInfo(lpManager).initialLpOwner(_lp) == msg.sender,"X SWAP Lp Vaults:
msg.sender is NOT the initial LP owner");
    require(IERC20(_lp).balanceOf(address(this))==IERC20(_lp).totalSupply(),"X SWAP Lp
Vaults: Other liquidity must be fully redeemed");
    _amount = iLpVaultInfo(lpManager).initLpAmount(_lp);
    IERC20(_lp).transfer(msg.sender,_amount);
    emit InitialLpRedeem( _lp, msg.sender, _amount);
}
```

Attach Scenario

1. Alice calls `unionswapinterface::xLpSubscribe()` to initialize a new pool. The `initialLpOwner` of LP in `unionswapinterface` contract is Alice and The `initialLpOwner` of LP in `xunionswaplpmanagement`contract is `unionswapinterface` .
2. After `lpTimeLimit` time, the pool has not been used.
3. Attacker can call `unionswapinterface::initialLpRedeem()` to get the LP token because `msg.sender` is `unionswapinterface` contract which is equal to `xunionswaplpmanagement::initialLpOwner()` When the `initialLpRedeem` function is executed in `xunionswaplpvaultsl` contract .
4. Alice loses the initLP token forever.

Recommendation

Add a permission control to the `xunionswapinterface::initialLpRedeem()` function.

Status

The team has resolved this issue in commit [fc592b8](fc592b8).

SALUS

4. CreatLpVault's approve may fail

Severity: Medium	Category: Business Logic
Target: - contracts/xunionswapvaults.sol	

Description

The `creatLpVault()` function will call the `approve()` function of `_tokens[0]` or `_tokens[1]` address.

xunionswapvaults.sol:L99-L113

```
function creatLpVault(address _lp,address[2] memory _tokens,uint8 lpCategory) external
onlyLpManager{
    ...
    IERC20(_tokens[0]).approve(lpManager, 999999999999999999999999999999999999 ether);
    IERC20(_tokens[1]).approve(lpManager, 999999999999999999999999999999999999 ether);
    ...
}
```

Attach Scenario
If the token0 is `COMP`, the `approve()` function will revert because `999999999999999999999999999999999999 ether` is greater than 2**96.

```
function approve(address spender, uint rawAmount) external returns (bool) {
    uint96 amount;
    if (rawAmount == uint(-1)) {
        amount = uint96(-1);
    } else {
        amount = safe96(rawAmount, "Comp::approve: amount exceeds 96 bits");
    }
    allowances[msg.sender][spender] = amount;
    emit Approval(msg.sender, spender, amount);
    return true;
}
function safe96(uint n, string memory errorMessage) internal pure returns (uint96) {
    require(n < 2**96, errorMessage);
    return uint96(n);
}
```

Recommendation

Use `type(uint256).max` instead of `999999999999999999999999999999999999 ether`.

Status

The team has resolved this issue in commit fc592b8.

SALUS

5. Use openzeppelin's ecdsa library instead of evm's ecrecover

Severity: Low	Category: Cryptography
Target: - contracts/xunionswappair.sol	

Description

The `permit()` function use `ecrecover()` to verify the signature. However, there is a Signature malleability risk for the evm's ecrecover. Even though there are no attack vectors in current implementation the highest security standards are always good.

xunionswappair.sol:L127-L139

```
function permit(address owner, address spender, uint value, uint deadline, uint8 v,
bytes32 r, bytes32 s) external {
    require(deadline >= block.timestamp, 'X SWAP Pair: EXPIRED');
    bytes32 digest = keccak256(
        abi.encodePacked(
            '\x19\x01',
            DOMAIN_SEPARATOR,
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline))
        )
    );
    address recoveredAddress = ecrecover(digest, v, r, s);
    require(recoveredAddress != address(0) && recoveredAddress == owner, 'X SWAP Pair:
INVALID_SIGNATURE');
    _approve(owner, spender, value);
}
```

Recommendation

Use the ECDSA from openzeppelin instead of ecrecover.

Status

The team has resolved this issue in commit fc592b8.

SALUS

6. Use safeTransfer()/safeTransferFrom() instead of transfer()/transferFrom()

Severity: Low	Category: Risky external calls

Target:
- contracts/xunionswapinterface.sol
- contracts/xunionswaplpmanagement.sol
- contracts/xunionswaplpvaults.sol
- contracts/xunionswapvaults.sol

Description

xunionswapinterface.sol:L90-L92

```
IERC20(TokensAddr[0]).transferFrom(msg.sender,address(this),_amountEstimated[0]);
IERC20(TokensAddr[1]).transferFrom(msg.sender,address(this),_amountEstimated[1]);
IERC20(_lp).transfer(msg.sender,IERC20(_lp).balanceOf(address(this)));
```

Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification:

```
Callers MUST handle false from returns (bool success). Callers MUST NOT assume that
false is never returned!
```

The agreement utilizes `` `transfer()/transferFrom()` `` in multiple instances.

Recommendation

Consider using the SafeERC20 library implementation from OpenZeppelin and call safeTransfer or safeTransferFrom when transferring ERC20 tokens.

Status

The team has resolved this issue in commit b9a0388.

SALUS

7. Not applicable for fee on transfer token

Severity: Low	Category: Business Logic

Target:
- contracts/xunionswapinterface.sol
- contracts/xunionswaplpmanagement.sol

Description

There are ERC20 tokens that charge a fee for each `transfer()` or `transferFrom()`, for example the PAXG token. As a result, when a token transfer occurs, the recipient's balance will be lower than expected.

xunionswaplpmanagement:L163-L183

```
function xLpRedeem(address _lp,uint _amountLp) external lock returns(uint[2] memory
_amount){
    ...
    IERC20(assetAddr[0]).transferFrom(xVaults,msg.sender,_amount[0]);
    IERC20(assetAddr[1]).transferFrom(xVaults,msg.sender,_amount[1]);

    //here need add info change
    ixVaults(xVaults).dereaseLpAmount(_lp, _amount,_amountLp);
    emit Redeem(_lp, msg.sender, _amountLp);
}
```

xunionswapinterface:L81-L102, 103-136, L140-L148, L149-L165

```
function xLpSubscribe(address _lp,uint[2] memory _amountEstimated) public
returns(uint[2] memory _amountActual,uint _amountLp) {
   ...
   IERC20(TokensAddr[0]).transferFrom(msg.sender,address(this),_amountEstimated[0]);
   IERC20(TokensAddr[1]).transferFrom(msg.sender,address(this),_amountEstimated[1]);
   IERC20(TokensAddr[0]).approve(xlpmanager, _amountEstimated[0]);
   IERC20(TokensAddr[1]).approve(xlpmanager, _amountEstimated[1]);
   (_amountActual,_amountLp) =
ixLpManager(xlpmanager).xLpSubscribe(_lp,_amountEstimated);
    ...
    }
}
function xLpSubscribe2(address _lp,uint[2] memory _amountEstimated) public payable
returns(uint[2] memory _amountActual,uint _amountLp) {
    ...
    if(TokensAddr[0] != wCFX){
IERC20(TokensAddr[0]).transferFrom(msg.sender,address(this),_amountEstimated[0]);
    }
    if(TokensAddr[1] != wCFX){
IERC20(TokensAddr[1]).transferFrom(msg.sender,address(this),_amountEstimated[1]);
    }
    IERC20(TokensAddr[0]).approve(xlpmanager, _amountEstimated[0]);
    IERC20(TokensAddr[1]).approve(xlpmanager, _amountEstimated[1]);
    (_amountActual,_amountLp) =
ixLpManager(xlpmanager).xLpSubscribe(_lp,_amountEstimated);
```


SALUS

```
    ...
    }
}
function xLpRedeem(address _lp,uint _amountLp) public returns(uint[2] memory _amount) {
    ...
    IERC20(_lp).transferFrom(msg.sender,address(this),_amountLp);
    IERC20(_lp).approve(xlpmanager, _amountLp);
    _amount = ixLpManager(xlpmanager).xLpRedeem(_lp,_amountLp);
    ...);
}
function xLpRedeem2(address _lp,uint _amountLp) public returns(uint[2] memory _amount) {
    ...
    IERC20(_lp).transferFrom(msg.sender,address(this),_amountLp);
    IERC20(_lp).approve(xlpmanager, _amountLp);
    _amount = ixLpManager(xlpmanager).xLpRedeem(_lp,_amountLp);
    ...
}
```

If the user enters the protocol with this type of token, the call to router in the above-mentioned function will revert because the contract has less than desc.amount of tokens at that point.

Recommendation

Consider calling `balanceOf()` to get the actual balances.

Status

The team has resolved this issue in commit 1c7b1ae.

SALUS

8. Missing events for functions that change critical state

Severity: Low	Category: Logging

Target:
- contracts/xunionswapfactory.sol
- contracts/xunionswapinterface.sol
- contracts/xunionswaplpmanagement.sol
- contracts/xunionswaplpvaults.sol
- contracts/xunionswapvaults.sol

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the xunionswapfactory contract, events are lacking in the privileged setter functions (e.g. `` `lpResetup()` ``, `` `rewardTypeSetup()` ``, `` `settings()` ``, `` `setPA()` ``, `` `acceptPA()` ``, `` `resetuplp()` ``).

In the xunionswapinterface contract, events are lacking in the privileged setter functions (e.g. `` `systemSetup()` ``, `` `transferLpSetter()` ``, `` `acceptLpSetter()` ``, `` `createPair()` ``, `` `xLpSubscribe()` ``, `` `xLpRedeem()` ``, `` `xexchange()` ``, `` `initialLpRedeem()` ``).

In the xunionswaplpmanagement contract, events are lacking in the privileged setter functions (e.g. `` `settings()` ``, `` `settingMinLpLimit()` ``, `` ` xLpInfoSettings()` ``, `` `exceptionTransfer()` ``, `` `xLpSubscribe()` ``, `` `xLpRedeem()` ``, `` `xexchange()` ``, `` `initialLpRedeem()` ``).

In the xunionswaplpvaults contract, events are lacking in the privileged setter functions (e.g. `` `systemSetup()` ``, `` `timeLimitSetup()` ``, `` `transferLpSetter()` ``, `` `acceptLpSetter()` ``, `` `initialTimeLimit()` ``, `` `exceptionTransfer()` ``, `` `setInitTime()` ``).

In the xunionswaplpvaults contract, events are lacking in the privileged setter functions (e.g. `` `systemSetup()` ``, `` `xInterfacesetting()` ``, `` `transferLpSetter()` ``, `` `acceptLpSetter()` ``, `` `exceptionTransfer()` ``, `` `creatLpVault()` ``, `` `increaseLpAmount()` ``, `` `dereaseLpAmount()` ``, `` `lpSettings()` ``, `` `addTokenApproveToLpManager()` ``).

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit [b9a0388](b9a0388).

SALUS

2.3 Informational Findings

9. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.0;
```

The XDex-protocol uses a floating compiler version ^0.8.0.

Using a floating pragma ^0.8.0 statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue in commit 5add31b.

SALUS

10. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization

Target:
- contracts/xunionswapcore.sol
- contracts/xunionswapinterface.sol
- contracts/xunionswapvaults.sol

Description

xunionswapcore.sol:L223

```
for(i=0;i<tokens.length-1;i++){
```

xunionswapinterface.sol:L216, L248

```
for(i=0;i<tokens.length-1;i++){
for(i=tokens.length-1;i>0;i--){
```

xunionswapvaults:L256

```
for(i=0;i<_exVaults.tokens.length-1;i++){
```

Memory reading saves more gas than storage reading multiple times when the state is not changed. So caching the storage variables in memory and using the memory instead of storage reading is effective.

Recommendation

Use suggestions to save gas.

Status

The team has resolved this issue in commit cc6886d.

SALUS

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 34369b2:

File	SHA-1 hash
contracts/xunionswapinterface.sol	00a764c9bbc2caa88aa14653468eb2cddbe1dadb
contracts/xunionswapvaults.sol	1faf39bbb1a6ce05c2988d8db493b9a52158fd0b
contracts/xunionswapcore.sol	33639ac1351ed5544dfa788118d4076bee9499ff
contracts/xunionswaplpmanagement.sol	bf5c5e28da59ed55bf84e7ec7eeaeb75e6f28a4c
contracts/xunionswappair.sol	f1f762010483bec60f9c06f82b07cacff3f2f873
contracts/xunionswapfactory.sol	8993d26c6f46b320ea64744abd130a349cd08c1f
contracts/xunionswaplpvaults.sol	9c365f37420c651c71cb871ff97127f4526278ec
contracts/periphery.sol	55299d6b091ca4bcd21be84593f725c9c9992a43
contracts/libraries/structlibrary.sol	23238d53b1d015fd0f0005c2f6322d6f54d4b19c

SALUS