SALUS SECURITY

OCT 2024

CODE SECURITY ASSESSMENT

CONFICTION

Overview

Project Summary

- Name: Confiction - XpsrDeposit
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/Confiction/xpsr-nft-deposit
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Confiction - XpsrDeposit
Version	v2
Type	Solidity
Dates	Oct 14 2024
Logs	Oct 11 2024, Oct 14 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	1
Total informational issues	1
Total	4

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	Users may not withdraw left funds	Medium	Business Logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Improper amount check in snapshotDeposit	Low	Business Logic	Resolved
4	Signatures may be reused	Informational	Business Logic	Acknowledged

2.2 Notable Findings

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

1. Users may not withdraw left funds	
Severity: Medium	Category: Business Logic
Target: - XpsrDeposit.sol	

Description

The depositors' balance will be reduced when the system takes a snapshot of their deposit. Once the `isWithdrawLive` feature is enabled, depositors can withdraw any remaining balance from their deposit.
The issue arises due to the use of the `checkSigned` modifier in the withdraw function. If the remaining balance falls below the `minimumDeposit`, depositors are unable to withdraw their remaining funds.
XpsrDeposit.sol:L155-L175

```
function withdraw(
    uint256 _nonce, uint256 _value, bytes32 _msgHash, bytes memory _signature
)
    external
    nonReentrant
    checkSigned(msg.sender, _value, _nonce, _msgHash, _signature)
{
    ...
}
```

XpsrDeposit.sol:L51-L62

```
modifier checkSigned(
    address _address, uint256 _value, uint256 _nonce, bytes32 _messageHash,
    bytes memory _signature
) {
    require(
        _value >= minimumDeposit,
          "Fund cant be lower than minimum deposit!"
        );
    …
}
```

Recommendation

Depositors should withdraw their left funds.

Status

This issue has been resolved by the team with commit be0b31a.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - XpsrDeposit.sol	

Description

In the `XpsrDeposit` contract, there exists a privileged role called `owner`. The `owner` has the authority to execute some key functions such as `setXpsrPrice`, `setDepositEnd`, `setSignerAddress`.

If the `owner`'s private key is compromised, an attacker could trigger these functions to block the key functions in the contract.

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. The team states that a multisig gnosis wallet will be used for the owner role.

3. Improper amount check in snapshotDeposit	
Severity: Low	Category: Business Logic
Target: - XpsrDeposit.sol	

Description

When the deposit is inactive, the owner can update the depositors' balances using the `snapshotDeposit` function. Within `snapshotDeposit`, it's necessary to verify that the depositors' balance is sufficient to cover the cost of the NFT.

The issue is that when the deposited amount matches the `nftPrice`, the system incorrectly assumes that the deposit is insufficient to cover the cost of the NFT.

XpsrDeposit.sol:L111-L134

```
function snapshotDeposit(SnapshotData[] calldata _listData)
    external
    onlyOwner
{
    require(
        !isDepositLive && block.timestamp > depositEnd,
        "Deposit is still live"
    );
    for (uint256 i = 0; i < _listData.length; i++) {
        uint256 currentDeposit = totalDeposit[_listData[i]._address];

        uint256 nftPrice = _listData[i]._totalSpot * xpsrPrice;
        require(
            totalDeposit[_listData[i]._address] > nftPrice,
            "Total deposit below nft price"
        );
        xpsrVault = xpsrVault + nftPrice;
        totalDeposit[_listData[i]._address] = currentDeposit - nftPrice;
    }
}
```

Recommendation

It should be acceptable that the deposited amount equals `nftPrice`.

Status

This issue has been resolved by the team with commit be0b31a.

2.3 Informational Findings

4. Signatures may be reused	
Severity: Informational	Category: Business Logic
Target: - XpsrDeposit.sol	

Description

Both the `xpsrDeposit` and `withdraw` functions require a signature from the `signerAddress`. However, the nonce in the signature data is not properly implemented in the contract, which may lead to the risk of signature reuse and potential security vulnerabilities.

By simply passing the same nonce, the signature can be reused, which may result in users depositing more ETH than the signer intended. If the `withdraw` function is open and the balance matches the `value` in the signature data, the signature used for the `deposit` function can also be reused for the `withdraw` function.

XpsrDeposit.sol:L91-L99

```
function xpsrDeposit(
    uint256 _nonce, bytes32 _msgHash, bytes memory _signature
)
    external
    payable
    checkSigned(msg.sender, msg.value, _nonce, _msgHash, _signature)
{
    ...
}
```

XpsrDeposit.sol:L147-L156

```
function withdraw(
    uint256 _nonce, uint256 _value, bytes32 _msgHash, bytes memory _signature
)
    external
    nonReentrant
    checkSigned(msg.sender, _value, _nonce, _msgHash, _signature)
{
    ...
}
```

Recommendation

We recommend storing the users' nonce instead of passing by parameters to implement proper nonce handling within the contract.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 848236d:

File	SHA-1 hash
XpsrDeposit.sol	5df5e035555f12c54b4d1e95903a1f10d1b35764